VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

DEPARTMENT OF COMPUTER SYSTEMS

ROZŠIŘUJÍCÍ MODUL S DOTYKOVÝM LCD DISPLEJEM PRO VÝUKOVOU PLATFORMU MINERVA

EXPANSION MODULE WITH TOUCH LCD DISPLAY FOR EDUCATIONAL MINERVA PLATFORM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. OLIVER NEMČEK**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. VÁCLAV ŠIMEK**

SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Nemček Oliver, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Rozšiřující modul s dotykovým LCD displejem pro výukovou platformu Minerva**
Expansion Module with Touch LCD Display for Educational Minerva Platform
Kategorie: Vestavěné systémy

Pokyny:

1. Detailně se zabývejte problematikou LCD displejů a dostupnými technologiemi pro realizaci interaktivní dotykové vrstvy.
2. Seznamte se s výukovou platformou Minerva. Pozornost zaměřte na použitý mikrokontrolér, hradlové pole FPGA a vývojové nástroje.
3. Navrhněte koncepci jednoduchého rozšižujícího modulu s vhodným typem dotykového LCD displeje a proveďte jeho technickou realizaci.
4. Navrhněte a implementujte low-level ovladač displeje (např. inicializace, smazání, rozsvícení bodu).
5. Navrhněte a implementujte grafickou knihovnu pro displej, která bude obsahovat běžné ovládací prvky a zobrazovače dat (tlačítko, posuvník, text, zaškrtávací políčko, přepínač, rozbalovací seznam...).
6. Vytvořte jednoduchou aplikaci, která bude prvky z knihovny vhodně demonstrovat.
7. Zhodnoťte dosažené výsledky pokuste se navrhnou případná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šimek Václav, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Cieľom tejto diplomovej práce je vytvoriť rozširujúci modul s TFT displejom, ktorý je vybavený dotykovou vrstvou, pre výukovú platformu Minerva. K tomuto modulu bude implementovaný nízkoúrovňový ovládač a knižnica pre grafické užívateľské rozhranie. Knižnica umožní jednoduchým spôsobom vytvoriť grafickú aplikáciu spustiteľnú na výukovom kite.

Abstract

The aim of this Master thesis is create an expansion module with TFT touch screen display for educational Mierva platform. Low level driver and graphical user interface library will be developed for this module. GUI library enables user to design an graphical application in easy way.

Kľúčové slová

LCD, TFT, FPGA, Kinetis K60, Minerva, LCD driver, rozširujúci modul

Keywords

LCD, TFT, FPGA, Kinetis K60, Minerva, LCD driver, expansion module

Citácia

NEMČEK, Oliver. *Rozšiřující modul s dotykovým LCD displejem pro výukovou platformu Minerva*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Rozšiřující modul s dotykovým LCD displejem pro výukovou platformu Minerva

Prehlásenie

Prehlasujem, že som túto diplomovu prácu vypracoval samostatne pod vedením pána Ing. Václava Šimeka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....

Oliver Nemček

24. mája 2017

Poďakovanie

Poďakovanie patrí vedúcemu práce, pánovi Ing. Václavovi Šimekovi, za všetky cenné rady, ktoré mi poskytol počas riešenia tejto práce.

Obsah

Kapitola 1

Úvod

Zobrazovacie zariadenia ako LCD TFT displeje sa stávajú nedeliteľnou súčasťou našich životov. Súvisí to hlavne s pokrokom a miniaturizáciou v technike. Aktuálny trend IOT, Smart zariadení, senzorov alebo monitorovacích prostriedkov si vyžaduje zobrazovať užívateľom informácie v komplexnej a človeku prirodzenej forme - graficky.

Úlohou rozširujúceho LCD displeja, ktorý bude predstavený v tejto práci, je dodať vizuálnu spätnú väzbu výukovej platforme Minerva. Tento kit je vybavený mikrokontrolerom Kinetis K60 a hradlovým poľom FPGA Xilinx Spartan-6. Disponuje dvomi rozširujúcimi konektormi, ktoré využijeme pre pripojenie externého modulu s TFT displejom. Zostavený expanzný modul obsahuje kapacitnú dotykovú vrstvu, ktorá sa využije na snímanie dotyku pre účely grafického užívateľského rozhrania.

Výuková doska s týmto displejom môže byť použitá všade tam, kde treba užívateľovi poskytovať spätnú väzbu vo forme obrazových dát. V spolupráci s externou SD kartou by sa na obrazovke mohlo zobrazovať video alebo obrázky. Otvoria sa nové možnosti integrácie dosky do iných projektov a produktov.

Táto práca je delená do troch logických celkov. V prvej časti sa výklad sústreďuje na hardvérovú realizáciu rozširujúceho modulu, výberom jednotlivých komponentov a návrhom plošného spoja. V ďalšej časti bude popísaný princíp nízkoúrovňového radiča a predstavené zapojenie modulu k FPGA čipu. Posledná časť je zameraná na jednoduchú knižnicu pre zobrazovanie grafického užívateľského rozhrania.

Dôraz je kladený na preskúmanie možnosti rýchleho zobrazovania rasterových dát. Komunikácia s displejom prebieha na 24-bitovom paralélnom rozhraní RGB, ktoré je určené pre zobrazovanie pohyblivého obrazu s frekvenciou až 60 Hz.

Riešenie tejto práce spája v sebe konštrukčný a programátorský problém. Najprv je potrebné zostaviť funkčnú hardvérovú realizáciu modulu. Následne na to vytvoriť v FPGA čipe radič rozhrania RGB. Posledná fáza rieši vytvorenie konceptu grafického užívateľského rozhrania, ktorého kód bude spustiteľný na mikrokontroleri.

V prvej časti sa práca zaoberá návrhom hardvéru rozširujúceho modulu. Najprv sú analyzované možnosti zapojenia externého modulu s obrazovkou. Popísaný je výber súčiastok

a návrh plošného spoja. V druhej časti práce je popísaná implementácia nízkoúrovňového ovládača obrazovky na jednotke FPGA. Komunikácia medzi mikrokontrolerom a FPGA čipom je zabezpečená pomocou FlexBus zbernice. Nad tým je implementovaná knižnica pre zobrazovanie grafického užívateľského rozhrania s podporou pre obvyklé komponenty.

V prílohe sa nachádzajú schémy rozširujúceho modulu a zdrojový kód demo-aplikácie, ktorá je postavená nad knižnicou a vhodne demonštruje jej možnosti a použitie.

Kapitola 2

Vývojová platforma

Predtým ako bude popísaný LCD modul, je nutné zoznámiť čítateľa s platformou Minerva. Cieľom tejto kapitoly je predstavenie cieľovej výukovej dosky z pohľadu dostupných mikročipov, rozširujúcich konektorov a komunikačných zberníc.

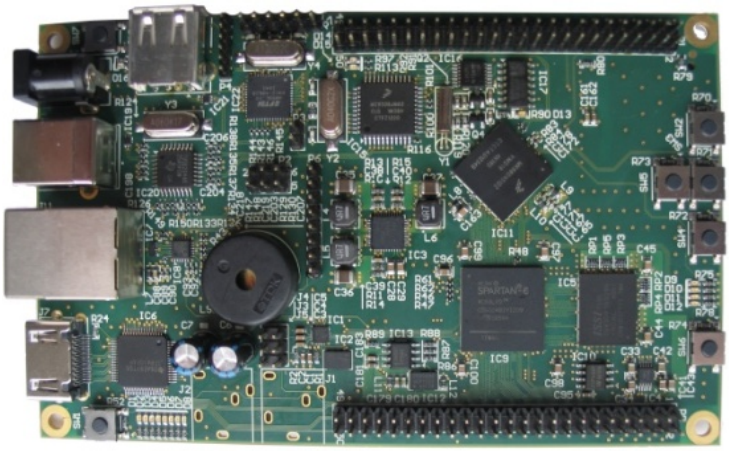
2.1 Minerva – výukový kit

Táto vývojová doska má potenciál stať sa nástupcom výukového kitu, ktorý je označovaný ako FITKit. Oproti svojmu predchodcovi Minerva disponuje moderným mikroprocesorom Kinetis rady K60 (MK60DN512VMD10) [5]. Jadro mikrokontroleru tvorí 32-bitový ARM Cortex-M4, výkonný jednojadrový ARM procesor s malou spotrebou. Určený je na spracovanie rôznych typov signálov. Mikrokontroler obsahuje množstvo modulov, ktoré rozširujú jeho použitie. RTC, Ethernet, CRC, ADC, DAC, USB, CAN, GPIO, UART a množstvo iných periférii je už integrovaných na čipe. Maximálna taktovacia frekvencia je 100 MHz. Mikroprocesor je pripojený k FPGA čipu pomocou rýchlej komunikačnej linky FlexBus. Okrem radičov periférií, ktoré sa na tomto mikrokontroleri nachádzajú, je na doske osadené USB s podporou OTG a Ethernet. Rovnako sa na doske nachádza HDMI port spolu s radičom, audio koder/dekoder a jeden DDR2 čip, pripojený priamo k FPGA. Rozhranie pre programovanie ladenie využíva pomocný mikrokontroler HCS08.

Zásadnou zmenou oproti FITKitu je absencia akéhokoľvek displeja. Predchodca disponoval jednoduchým 16x2 znakovým displejom. Ďalším limitujúcim faktorom je fakt, že zatiaľ neexistuje dostatok vysokoúrovňových knižníc, ktoré by študenti mohli používať pre experimentovanie s vývojovou doskou.

Webová stránka [2] popisuje jednotlivé každú komponentu platformy, Bohužiaľ, mnoho odkazov nie je funkčných a pravdepodobne stránka už nie je udržiavaná. Niektoré odkazy smerujú na nesprávne dokumenty.

V súčasnej dobe sa rôzne mikropočítačové systémy tešia veľkej obľube, hlavne kvôli ich cene a pomerne vysokému výkonu ktorý poskytujú. Spomeňme napríklad jedného z ich zástupcov Raspberry Pi, ktorého cena nepresahuje 35 dolárov. K septembru 2016 sa predalo



Obr. 2.1: Minerva, vrchná strana, prebraté z [2].

viac ako 10 miliónov kusov[1]. Vývoj software a hardware zastrešuje obrovská komunita vývojárov. Druhým známym zástupcom malých jednoúčelových hobby zariadení je Arduino. Táto platforma sa teší vysokému počtu knižníc, ktoré sú k dispozícii pre periférie všetkých druhov. Cena senzorov a iných modulov je veľmi priaznivá. Aktuálny trend IOT zariadení len umocňuje dopyt po lacných mikročipoch s malou spotrebou.

Naša cieľová platforma používa mikroprocesor, ktorý je priemyselne zameraný. Podľa dokumentov od výrobcu je určená pre spracovanie rôznych typov signálov. Najväčší dôvod, ktorým sa Minerva odlišuje od lacných alternatív je prítomnosť FPGA čipu, ktorý je na integrovaný priamo na vývojovej doske. FPGA je čip založený na technike hradlových polí, ktoré je možné prekonfigurovať a realizovať vlastné zapojenie dynamicky, na softvérovej úrovni. Ideou rekonfigurovateľného hardvéru je vysoká univerzálnosť, škálovateľnosť, a možnosť vytvoriť hardvérovo akcelerovanú výpočetnú jednotku na mieru.

V nasledujúcich podkapitolách sa pokúsime popísať rozdelenie zodpovednosti jednotlivých častí vývojovej dosky z hľadiska povahy tejto práce. V prvom rade je dôležité si uvedomiť, že pre zapojenie LCD modulu nemôže byť použitý iba mikrokontroler. Dôvodom je malý počet pinov, ktoré sú vyvedené na externú päticu. Rovnako tak vzniká požiadavka na konštantné prenášanie obrazových dát na LCD panel. Potreba vysokého výkonu pri nepretržitom prenose dát by zaberala takmer všetok čas na mikroprocesore. Vzhľadom na tento fakt je nutné presunúť niektoré činnosti na FPGA čip. Rozdelenie zodpovednosti bude popísané v ďalších kapitolách.

2.2 Procesor

Hlavný mikroprocesor Kinetis K60 implementuje nízkoúrovňový prístup k LCD displeju. Bude vykresľovať užívateľské rozhranie do obrazovej pamäti. V prípade potreby môže ras-

[1] Zdroj: https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit

terizovať elementy grafického užívateľského rozhrania. Obrazová pamäť sa v grafickej terminológii nazýva tzv. framebuffer. Je to označenie pre alokované miesto v pamäti, kde sa nachádzajú obrazové dáta vo vhodnej reprezentácií. Prístup do grafickej pamäte by mal byť rýchly, aby mikroprocesor vedel obnovovať obrazovku dostatočne rýchlo. Okrem zápisu je výhodné, aby sa dali obrazové dáta aj čítať.

Jeden z problémov, ktorý treba vyriešiť je, že procesor nemá dostatočnú pamäťovú kapacitu pre uloženie celého framebufferu. Pre hrubú predstavu koľko pamäti je potrebnej pre framebuffer môžeme použiť jednoduchú rovnicu. Predpokladajme, že rozlíšenie displeja je 320x240 bodov. Bitová hĺbka je nastavená na 8 bitov pre každý kanál RGB. Potom potrebná pamäť je 320x340x8x3 bitov. Výsledkom je 225 kB, ale k dispozícii je iba RAM o veľkosti 128 kB. Tento problém sme sa rozhodli vyriešiť pomocou podporného obvodu realizovaného na FPGA.

Najdôležitejší proces, ktorý bude vykonávaný na mikroprocesore je kód pre grafické užívateľské rozhranie. Z hľadiska implementácie sa môžeme inšpirovať niektorým z existujúcich riešení, ako vytvoriť jednoduché a efektívne rozhranie s minimálnym dopadom na výkon a pamäťové nároky. Využitie objektového jazyka dokáže problém dekomponovať a tým znížiť jeho náročnosť na implemenáciu. Inšpirujeme sa existujúcimi knižnicami, aby sa minimalizovala náročnosť používania knižnice pre nového užívateľa.

Pre kompletnosť uveďme vzťah mikroprocesoru k dotykovému panelu. Najrozumnejším návrhom je integrovať podporu dotykovej spätnej väzby priamo do interných štruktúr GUI. Najlepšia varianta pre komunikačné rozhranie dotykovej plochy je, aby bolo kompatibilné s jedným z komunikačných modulov, ktoré sa nachádzajú v mikrokontroleri. Vhodným rozhraním môže byt napríklad SPI alebo I2C.

2.3 FPGA

Úloha hradlového poľa FPGA vrámci tejto práce je realizovať spojnicu medzi LCD modulom a mikrokontrolerom. FPGA musí implementovať fundamentálnu časť rozhrania pre komunikáciu s modulom pre RGB rozhranie. Jeden funkčný blok bude realizovať generovanie RGB signálov spolu s prenosom dát. Druhý funkčný blok bude komunikovať s externou pamäťou typu DDR2. Tretí funkčný blok bude implementovať riadiacu komunikáciu s radičom LCD obrazovky. Rozhranie RGB je náchylné na správne časovanie signálov. Generovanie hodinového signálu v FPGA je možné riešiť viacerými spôsobmi. Kritický je prenos hodinového signálu do ostatných častí realizovaného obvodu, pretože môže dochádzať k drobným nepresnostiam a spomaleniam signálu kvôli bufferovacím registrom. Je teda potrebné zabezpečiť správny zdroj hodinového signálu a rovnako aj jeho rozvod do ostatných častí realizácie. Sú to požiadavky, ktoré sú závislé na použitom FPGA čipe.

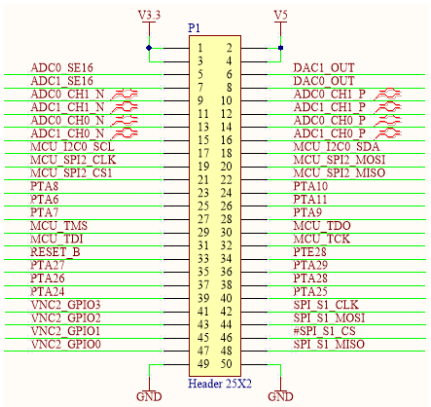
Signalizácia rozhrania RGB bude popísaná v samostatnej kapitole. Počet vodičov, ktoré budú prenášať obrazové dáta je 24. Každá farebná zložka RGB modelu má hĺbku 8 bitov. Podporná signalizácia spolu s riadením podsvietenia obsahuje ďalších 9 vodičov. Vyčerpáme

tak podstatnú časť voľných kontaktov, ktoré sa nachádzajú na rozširujúcej pätici. Riadenie podsvietenia je typicky realizované pomocou PWM signálu.

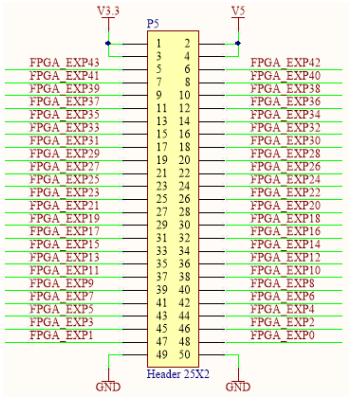
Ako bolo spomenuté, FPGA čip je pripojený priamo k DDR2 modulu o veľkosti 512 Mb. Pamäť RAM sa môže použiť ako miesto pre uloženie obrazovej pamäti – framebufferu. Problémom môže byť spojazdnenie komunikácie medzi DDR2 pamäťou a VHDL kódom, ktorý bude realizovaný na FPGA čipe. Prístupová doba do pamäti sa pohybuje v rádoch stoviek nanosekúnd. Tento čas je možné eliminovať napríklad použitím malej vyrovnávacej pamäte, do ktorej sa naraz načíta niekoľko bajtov.

Podľa dokumentácie podporuje použitý FPGA čip pripojenie k DDR2 pamäti pomocou dedikovanej periférie ktorú stačí inštanciovať v programovom kóde. Použitím dedikovaných komponentov šetríme miesto v FPGA čipe pre realizáciu iných obvodových zapojení. Ďalšie informácie o pripojení RAM pamäte budú uvedené v implementácii nízkoúrovňového radiča.

2.4 Rozširujúce rozhranie



(a) Rozširujúci konektor pripojený na Kinetis.



(b) Rozširujúci konektor pripojený na FPGA.

Obr. 2.2: Rozširujúce konektory na doske Minerva, prebrané z dokumentácie v programe Altium

Na platforme sa nachádzajú dva expanzné konektory (pätice) umiestnené na okrajoch vývojovej dosky. Schéma zapojenia prebratá z dokumentácie sa nachádza na obrázku č. 2.2. Prvá pätica (obrázok č. 2.2b) je pripojená k FPGA čipu, druhá (obrázok č. 2.2a) je pripojená k mikroprocesoru. Pre pripojenie nášho modulu musíme využiť obe pätice vzhľadom na rozdelenie zodpovedností popísanom v predošlých kapitolách. Okrem toho sú na obe pätice vyvedené napájacie VCC piny pre +3 a +5 voltov a uzemnenie GND. V prípade konštrukcie dosky rozširujúceho modulu je nutné dodržať presné rozostupy oboch pätíc a rozostup jednotlivých pinov 2,54 mm. Pre pripojenie dotykovej vrstvy sú na druhú päticu vyvedené konektory rozhrania I2C a SPI.

Bežnou praxou je vyrábať rozširujúce moduly a karty tak aby bolo možné ich viacnásobné zapojenie. Realizuje sa to napríklad použitím zberníc, ktoré umožňujú zdieľaný prístup viacerých zariadení. Iná alternatíva je vyviesť potrebné piny na telo modulu v rovnakom poradí ako je na vyvojovej doske. V tomto prípade sa ďalšie moduly na seba môžu zapájať a vytvára sa tak zásobníková architektúra. Pomyselný vrch zásobníka je v tomto prípade posledný modul, ktorý ma svoj rozširujúci konektor prázdny.

V prípade expanzného modulu, ktorý obsahuje LCD displej je zbytočné uvažovať o vyvedení nepoužitých kontaktov na vrchnú stranu modulu. LCD displej sa bude vždy nachádzať úplne navrchu pomyselnej veže modulov, inak by jeho zapojenie v sústave nemalo význam. V prípade potreby sa môže vyrobiť špeciálny adaptér, ktorý by sa zapájal pod náš modul. Adaptér by vyviedol nepoužité konektory na dostupné miesto.

Kapitola 3

Návrh a konštrukcia modulu

Predtým ako začneme navrhovať modul vo forme elektrických schém, musíme najprv vybrať komponenty, z ktorých sa bude modul skladať. V tejto fáze hrali najväčšiu rolu veľmi cenné rady vedúceho práce. Následkom absencie vlastných skúsenosti bolo, že táto fáza bola najpomalšia, čo sa týka celkového času stráveného riešením tejto práce.

Základné parametre modulu sú nízka cena, rozumná veľkosť obrazovky, rozlíšenie a vysoká rýchlosť zobrazovania. Vzhľadom na rozmery vývojovej dosky Minerva, pozíciu osadených súčiastok a výškový profil súčiastok sme sa rozhodli vytvoriť modul, ktorý sa bude zapájať z vrchnej pozície na rozširujúce konektory. V prípade väčšieho rozmeru modulu LCD by sa musela vytvoriť doska zakrývajúca celú plochu Minervy. Doska by musela byť umiestnená vyššie od základne, preto by bolo nutné použiť dištančné stĺpiky.

Na základe odporúčaní sme ako hlavný navrhový program použili Altium Designer vo verzii 17. V tomto programe bola vytvorená schéma dosky Minerva. Projektové súbory boli poskytnuté vedúcim práce hlavne pre zaistenie správnych fyzických rozmerov pinov.

3.1 Výber LCD modulu

Na základe poskytnutých rád sme zvolili el. obchody renomovaných dodávateľov Mouser a Farnell. Vzhľadom na rozmery vývojovej dosky, cca 127x76 mm, sme sa rozhodli hľadať LCD modul o veľkosti približne 3,5” s rozumnou cenou a s podporou 24-bitového paralelného rozhrania RGB. Mnoho produktov obsahuje integrovaný radič displeja, ktorý v sebe integruje aj obrazovú pamäť. Na druhú stranu, pripojenie k radiču je obyčajne realizované pomocou SPI (prípadne I2C) rozhrania a zďaleka nedosahuje takú výkonnosť, akú požadujeme. Kvôli predpokladu, že obrazová pamäť bude uložená v DDR2 čipe sa nemusíme obmedzovať na moduly, ktoré majú integrovaný kontroler.

V pomere ceny a výkonu sme zvolili bežne dostupný TFT LCD modul v cene do 50 dolárov. Označenie je `NHD-3.5-320240MF-ATXL#-CTP-1` od výrobcu Newhaven Display International, Inc. Displej má rozlíšenie 320x240 bodov a veľkosť uhlopriečky 3,5” [13]. Displej disponuje integrovaným radičom (NV3035C) [15], ktorý sa stará o riadenie TFT tranzisto-

rov. Video signál sa prenáša pomocou 24-bitového rozhrania RGB. Konfigurácia kontroleru prebieha po neštandardnej sériovej linke realizovanej 3 vodičmi. Iné možnosti pripojenia na daný radič neexistujú. Displej obsahuje dotykový vrstvu realizovaný vo forme kapacitnej vodivej plochy. Okrem toho obsahuje aj radič dotykovej vrstvy so štandardným rozhraním I2C.

Pre prípad potreby sme vybrali ešte jeden modul od rovnakého výrobcu s označením `NHD-3.5-320240MF-ATXL-T-1` [14]. Jeho zobrazovacia časť je rovnaká, ale líši sa použitou technológiou dotykovej vrstvy. Druhý modul disponuje rezistívnym povrchom. Výhodou tejto technológie je že reaguje na tlak – môže sa použiť ľubovolný predmet. Nevýhodou je, že k detekcii sa musí fyzicky pritlačiť vrchná dotyková vrstva k spodnej vrstve. Ľahšie dôjde k mechanickému poškodeniu tejto časti. Výhodou kapacitnej vrstvy je možnosť registrovať viacnásobný dotyk. Nevýhodou je, že registrovaný predmet musí byť vodivý. V závislosti na aplikácii sa výhody a nevýhody môžu striedať. Napríklad, kapacitná vrstva stráca svoju nevýhodu ak sa v danej aplikácii očakáva, že predmet, ktorý sa dotýka vrstvy je ľudský prst. Bolo by zaujímavé integrovať podporu viacnásobného dotyku do implementácie grafického rozhrania, umožnilo by to ovládanie pomocou gest, ktoré poznáme z mobilných aplikácií.

Pre modul s rezistívnou plochou je vhodné integrovať na dosku nášho modulu kontroler. Odtieni to náročnosť detekcie dotyku a dovolí to zamerať sa na podstatnú vlastnosť – pozíciu dotyku. Náročnosť výpočtu dotyku sa významne zvyšuje v prípade, že displej nie je osadený súbežne s dotykovou vrstvou. Súvisí to s prepočítavaním súradníc v pohyblivej desatinnej čiarke a potrebe vykonať kalibráciu. Popis vlastností externého radiča dotykovej vrstvy popíšeme neskôr.

Displej s kapacitnou vrstvou už obsahuje integrovaný kontroler s rozhraním I2C, to znamená že žiadne ďalšie súčasti nie sú potrebné. Pri použití kontroleru sa mikroprocesor odtieni a abstrahuje od použitej technológie snímania dotyku.

Rozhranie modulu je vyvedené na tzv. FFC (Flexible Flat Cable). Konektor obrazovej časti obsahuje 54 pinov. Dotyková časť sa líši použitou technológiou snímania dotyku a obsahuje 4 piny, v prípade rezistívnej plochy, alebo 6 pinov v prípade kapacitnej plochy.

Drobnou výhodou rezistívnej vrstvy, ktorá je dodávaná priamo s displejom je, že bola osadená rovnobežne s hranami displeja, preto by nemala byť potrebná zložitá kalibrácia dotykových súradníc.

Aby sa dal modul rozumným spôsobom zapojiť na plošný spoj výrobca odporúča v dokumentácii [13, 14] konkrétne výrobné čísla Molex konektorov. Tieto súčiastky sa pripojili k finálnej objednávke.

Obrázok č. 3.1 zobrazuje zakúpený LCD modul.

3.2 Súčiastky

Väčšina LCD displejov potrebuje samostatné konektory na napájanie. Vybratý modul nie je výnimkou. Na napájanie podsvietenia je potrebné dodávať 19,2 V pri prúde 18 mA. Vzhľa-

Obr. 3.1: Obrázok zakúpeného modulu, zdroj: www.newhavendisplay.com.

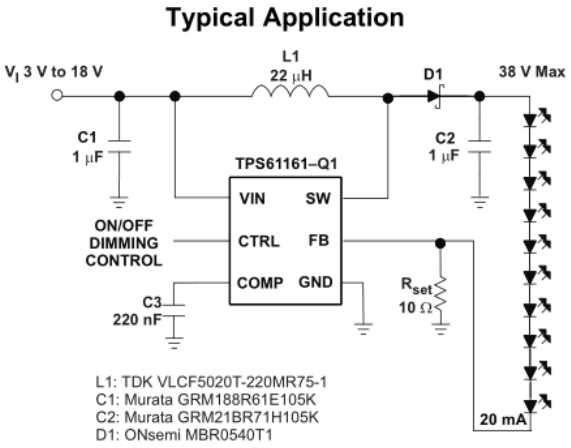
dom na to, že dostupné napätia na externom konektore na doske Minerva sú obmedzené na 3 V alebo 5 V je patrné, že na náš modul sa musí integrovať tzv. LED Driver. Táto súčiastka pracuje na princípe spínaného zdroja, ktorý je doplnený o obvod, ktorý limituje prúd.

Po krátkom prieskume existujúcich riešení pre riadenie LED diód na internete sme sa rozhodli pre výrobok spoločnosti Texas Instruments. `TPS61161` [19] je veľmi malý LED Driver s možnosťou riadenia pomocou PWM signálu. Vyrába sa v púzdre DRV6. Výhodou je vysoká efektivita a rozmedzie vstupného napätia, ktoré sa nachádza v rozsahu 2,7 V až 18 V. Na výstup sa dá zapojiť maximálne 10 LED diód, každá s napätím 3,6 V.

Ďalšie súčasti obvodu, ktorý sa stará o napájanie podsvietenia sú cievka, kondenzátory, Schottkyho dioda a rezistor R_{Set}. Prúd, ktorý preteká obvodom je limitovaný týmto rezistorom. Interný komparátor porovnáva napätie na odpore R_{Set} s referenčným napätím 200 mV. Podľa výsledku porovnania sa iný obvod stará o zvyšovanie/znižovanie napätia tak aby výsledné napätie na rezistore bolo 200 mV. Podľa dokumentácie má byť napájací prúd podsvietenia na úrovni 18 mA. To znamená, že hodnota odporu musí byť približne 11 Ohmov. Výpočet je priamou aplikáciou Ohmového zákona.

Všetky súčiastky sú znázornené na typickom zapojení driveru TPS61161 na obrázku č. 3.2.

Drobnou nevýhodou vybratého LCD Driveru je jeho malá veľkosť. Okrem toho všetky ostatné súčiastky sú zvolené tak, aby boli v štandardných rozmeroch „0603“ v imperiálnej



Obr. 3.2: Typické zapojenie obvodu s TPS61161 [19]

miere. V metrickej miere je to 1,6 mm x 0,8 mm. To môže robiť problém pri ručnom osadzovaní súčiastok na dosku plošného spoja.

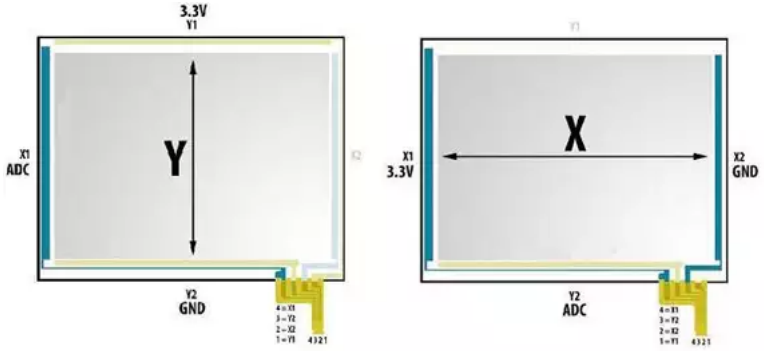
3.2.1 Dotyková vrstva

V minulosti sa najčastejsie používala dotyková vrstva pomocou rezistívnych plôch. V dobe nástupu „Smart telefónov“ sa trend otočil v prospech kapacitnej vrstvy. Ako už bolo spomenuté, výhody a nevýhody oboch variánt sú závislé na aplikačnom použití.

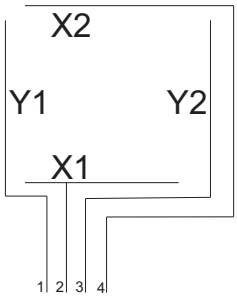
Z technického hľadiska je rezistívna vrstva založená na princípe odporového deliča. Skladá sa z dvoch vrstiev priehľadného vodivého materiálu. Medzi vrstvami je malá vzduchová kapsa. Vrstvy zvierajú uhol 90 stupňov. Pri aplikovaní tlaku, napríklad pri dotyku predmetu, dôjde k mechanickému prehnutiu vrchnej vrstvy, ktorá sa spojí s tomto mieste so spodnou vrstvou. Následne sa zmeria veľkosť napätia v smeroch X a Y, z ktorých sa algoritmicky vypočítajú súradnice dotyku. Prehľadný obrázok popisujúci princíp merania pozície sa nachádza na obrázku č. 3.3. Vzhľadom na konduktívny charakter ploch a fyzikálnych princípov, zmena napätia by mala byť lineárne závislá na zmene miesta dotyku.

Displej s rezistívnou plochou má podľa dokumentácie usporiadané meracie elektródy podľa obrázku č. 3.4. Toto usporiadanie je dôležité z hľadiska zapojenia vývodov konektora na kontroler. Typicky má rezistívna vrstva matnú povrchovú úpravu. Táto členitá štruktúra ovplyvňuje prepúšťané svetlo a obraz sa javí mierne rozmazaný.

Oproti tomu, pri kapacitnej dotykovej ploche nedochádza k žiadnemu mechanickému prehnutiu materiálu. Detekcia je založená na meraní zmeny elektrostatického poľa. Vodivý materiál, napríklad prst, pri priblížení k tejto vrstve zmení jej elektrostatické pole, ktoré sa dá zmerať ako zmena v kapacite. Nespornou výhodou tejto technológie je podpora pre viacnásobné miesta dotyku. V interakcii s človekom to umožňuje využívať rôzne gestá, ktoré sú intuitívne a dobre zapamätateľné.



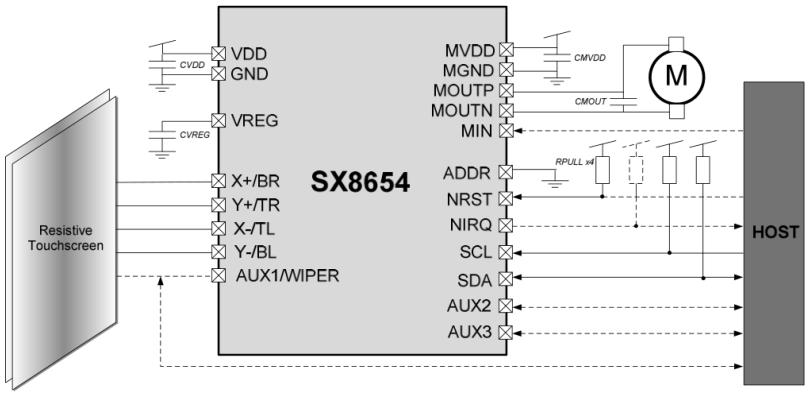
Obr. 3.3: Princíp určenia pozície na rezistívnom dotykovom paneli. Prevzaté zo stránky www.digikey.com



Obr. 3.4: Usporiadanie elektrod na rezistívnom dotykovom module.

Nami vybratý rezistívny displej nie je osadený kontrolerom dotykovej plochy. K objednávke sme pripojili aj tento čip. V spolupráci s vedúcim diplomovej práce sme zvolili integrovaný obvod s označením `SX8656` [18]. Tento čip je možné pripojiť na klasické 4-konektorové zapojenie. Okrem toho ponúka aj pripojenie 5-pinového zapojenia, ktoré však v našom prípade nebude použité. Komunikačný protokol je I2C, rovnako ako pri kapacitnej dotykovej vrstve.

Podľa dokumentácie [18] je možné jednoduchým trikom použiť vrchnú stranu dotykovej vrstvy pre účely detekcie priblíženia tzv. „proximity sensing“. V princípe sa jedná o použitie vrchnej strany dotykovej plochy pre meranie zmeny v elektrostatickom poli, rovnako ako v prípade kapacitnej technológie. Spodná strana plochy sa využíva ako uzemnenie. V tom prípade je plocha schopná detekcie len v jednom smere a efektívne je tak zabránené vzniku falošných detekcií za displejom. Túto vlastnosť dostávame v prípade použitia tohoto čipu zadarmo. Pre každý prípad sa výrobca rozhodol dať užívateľom možnosť pripojiť externé kapacitné elektródy. Nastavením kontrolných registrov je možné vybrať si, ktorá varianta bude použitá. Z pohľadu riešenia užívateľského rozhrania môže mať tento senzor zaujímavé uplatnenie napríklad pri zvýšení podsvietenia displeju, resp. zobudení mikrokontroleru pri detekcii blízkeho objektu.

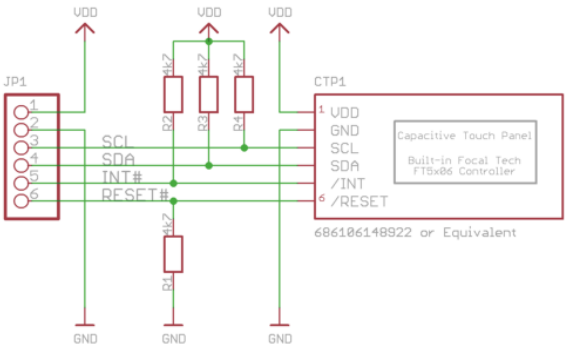


Obr. 3.5: Typické zapojenie dotykových kontrolerov z rodiny SX865x. [18]

Elektrické zapojenie a podporné súčiastky, ktoré musia byť v obvode je možné vidieť na obrázku č. 3.5.

Displej s kapacitnou plochou má na ohybnom pásiku integrovaný radič `FocalTech FT5216` [3]. Podľa špecifikácie má tento typ kontróleru zvládnuť až 5 simultánnych dotykov. Komunikácia s ním prebieha pomocou rozhrania I2C.

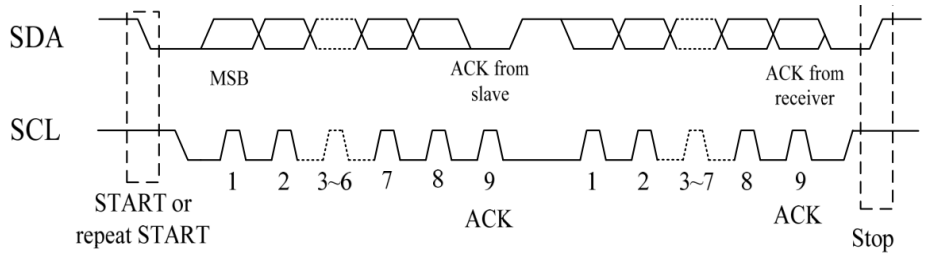
Obidva kontrolery dotykovej plochy využívajú pre komunikáciu port I2C a dva rozširujúce signálové vodiče. Základná schéma komunikácie sa nachádza na obrázku č. 3.7. Logické hodnoty hodinového signálu a dátového vodiča I2C sú aktívne v napätí 0 V. Preto sa typicky na I2C linku zapájajú tzv. pull up rezistory. Zvolili sme rovnaké zapojenie ako modul `CTP6` od výrobcu NewHavenDisplay [12]. Priložená schéma modulu sa nachádza na obrázku č. 3.6



Obr. 3.6: Zapojenie rezistorov na module CTP6 pre rozhranie I2C.

Zapojené resistory na I2C vodičoch `SDA` a `SCL` sú pripojené na zdroj napätia 3,3 V. Rozhranie sa tak bude nachádzať v nečinnom režime – stav logickej nuly. Pokiaľ jedna komunikujúca strana uvedie vodič na hodnotu 0 V, potom nastane drobný prúdový odber na danom rezistore. Použitím odporov o hodnote 4700 kΩ a napätím 3,3 V dôjde k prúdovému odberu približne 0,7 mA.

Ďalšie dva signálové vodiče sú označené ako `RESET` a `INT`. Prvý z nich sa používa ako externý resetovací signál dotykového kontroleru. K aktivácii resetu dochádza pri privedení 0 V na tento vodič. Pripojený „pull down“ odpor uvádza čip do resetu, pokiaľ sa na tento vodič nepripojí napätie približne 3 V. Druhý signál `INT` sa využíva ako indikátor prerušenia. Informuje o tom, že došlo k určitej udalosti na dotykovej vrstve, napríklad bolo detekovaný dotyk alebo pohyb pera. Rovnako ako ostatné signály je prerušenie aktívne v hodnote 0 V. Pripojením rezistoru k napájaciemu napätiu uvádzame tento signál do kľudového stavu. Na strane mikrokontroleru sa tento signál môže zapojiť na kontakt, ktorý podporuje prerušenia. V tom prípade nebude treba vykonávať tzv. „polling“, teda nepretržité testovanie hodnoty signálu. Externým prerušením sa dá dosiahnuť efektívna detekcia udalostí s malou odozvou.



Obr. 3.7: Časový graf protokolu I2C podľa zdroja[3].

3.3 Popis rozhrania zvoleného LCD modulu

Pre účely návrhu plošného spoja je potrebné poznať, akým spôsobom sa zapája fyzické rozhranie LCD modulu. Následne na to popíšeme signalizáciu rozhrania pre zobrazovanie obrazu. Zvolili sme modul s kapacitnou technológiou pretože už priamo obsahuje integrovaný radič dotykovej vrstvy.

3.3.1 Fyzické rozhranie

LCD panel, ktorý je na obrázku č. 3.1 má konektory vyvedené na ohybnom FFC pásiku. Šírka konektora, vzdialenosť jednotlivých pinov, ich dĺžka a rozostup sú uvedené v produktovej špecifikácii [13]. Aby sme nemuseli pracne hľadať správnu päticu konektora, výrobca do dokumentácie uviedol číslo produktu od spoločnosti Molex.

Fyzicky sa na konektore nachádza 54 kontaktov. Ich popis podľa dokumentácie sa nachádza na obrázku č. 3.8.

Piny 1 – 4 sú vyhradené pre napájanie LED diód, ktoré tvoria podsvietenie displeja. Náš radič podsvietenia sa bude pripája na tieto konektory. Napájací prúd nesmie prekročiť 20 mA, inak hrozí zničenie osvetľovacích diód. V návrhu plošného spoja sa musí počítať s tým, že napájacie vedenie má hodnotu napätia 19 V.

Pin No.	Symbol	External Connection	Function Description
1-2	LED_K	Power Supply	Backlight Cathode (Ground)
3-4	LED_A	Power Supply	Backlight Anode (18mA @ 19.2V)
5-7	NC	-	No Connect
8	RSTB	MPU	Active LOW Reset signal
9	SPENB	MPU	Active LOW Serial Chip Select signal
10	SPCK	MPU	Serial Clock signal
11	SPDA	MPU	Serial Data signal
12-19	B0-B7	MPU	Blue Data signals
20-27	G0-G7	MPU	Green Data signals
28-35	R0-R7	MPU	Red Data signals
36	HSD	MPU	Horizontal (Line) Sync signal
37	VSD	MPU	Vertical (Frame) Sync signal
38	CLKIN	MPU	Dot Clock signal
39-40	NC	-	No Connect
41-42	VDD	Power Supply	Supply Voltage for LCD and logic (3.3V)
43-51	NC	-	No Connect
52	DEN	-	Data Enable signal (No Connect)
53-54	GND	Power Supply	Ground

Obr. 3.8: Zapojenie 54-pinového konektora na module LCD [13].

Piny 9, 10 a 11 slúžia pre tzv. 3-wire SPI rozhranie. Vo svojej podstate ide o variantu SPI, ktorá využíva zdieľaný vodič pre obojsmernú komunikáciu. Duplexná komunikácia v SPI je serializovaná, pretože sa jeden signálový vodič odobral. Toto rozhranie sa používa pre konfiguráciu interných registrov v LCD radiči, ktorý sa nachádza na ohybnom pásiku. Pravdepodobne je možné toto rozhranie úplne vynechať a ponechať kontroler v základnom režime. Avšak, pre optimálne vlastnosti reprodukovaného obrazu nám výrobca odporúča nastaviť dva konkrétne registre.

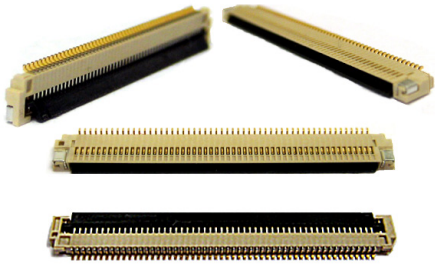
Na pin 8 je vyvedený externý reset. Je aktívny v hodnote 0 V a pokiaľ je v tomto stave, potom sa na displeji nič nezobrazuje, pretože radič LCD nie je v aktívnej prevádzke. Pre zahájenie štartovacej sekvencie radiča sa musí tomuto signálu priradiť hodnota 3,3 V.

Piny 12 – 35 vedú signál, ktorý určuje farbu pixelu. Tvoria ho červená (R), zelená (G) a modrá (B) farba. Každá farba má hĺbku 8 bitov. To znamená, že počet všetkých zobraziteľných farieb je $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$. Toto rozhranie používa paralelny prenos a preto môže byť náchylné na bitovú chybu pri vysokých taktovacích frekvenciách.

Piny 36 – 38 tvoria riadiacu signalizáciu RGB rozhrania. Toto rozhranie bude popísané neskôr.

Posledné piny 41–42 a 53–54 slúžia ako napájanie pre radič LCD. Maximálny prúdový odber pri 3,3 V je podľa dokumentácie 40 mA.

Použitým fyzickým konektorom je `51296-5494` od spoločnosti Molex, obrázok č. 3.9. Konektor pre dotykový panel je `52271-0679`. Je to 6-pinový konektor. Dôležitou poznámkou je že FFC pásik pre dotykové rozhranie je kratší približne o 0,5 cm ako pásik pre obrazové rozhranie. Táto informácia je dôležitá z hľadiska osadenia súčiastok na dosku plošného spoja.



Obr. 3.9: 54-pinový konektor Molex 51296-5494, zdroj: stránka NewHaven Display.

Popis signálov v 6-pinovom konektore sa nachádza na obrázku č. 3.10. Signály už boli popísané v časti 3.2.1. Okrem napájania sa tu nachádzajú vodiče zbernice I2C a resetovací signál. Signál *INT* indikuje, že došlo k udalosti spojenej s dotykom alebo posunom pera na obrazovke.

Pin No.	Symbol	External Connection	Function Description
1	VDD	Power Supply	Supply voltage for Logic (3.0V)
2	VSS	Power Supply	Ground
3	SCL	MPU	Serial I2C Clock (Requires pull-up resistor)
4	SDA	MPU	Serial I2C Data (Requires pull-up resistor)
5	/INT	MPU	Interrupt signal from touch panel module to host
6	/RESET	MPU	Active LOW Reset signal

Obr. 3.10: 6-pinový konektor Molex 51296-0679, zdroj: NewHaven Display.

Napájacie napätie je 3,3 V. Prúdový odber pri tomto napätí je maximálne 6 mA.

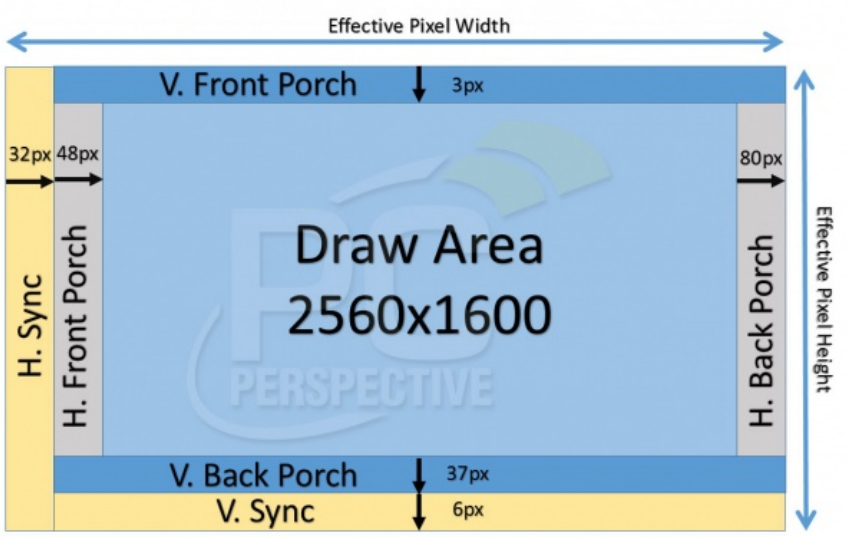
3.3.2 Rozhranie RGB

Rozhranie, ktoré riadi zobrazovanie farebných pixelov na TFT paneli sa nazýva *RGB Interface*. Môžeme ho charakterizovať ako paralelnu synchrónnu linku. Využíva sa hlavne tam, kde je na prvom mieste požiadavka na rýchlosť zobrazovania dát. V nasledujúcom výklade vychádzame hlavne z dokumentu [16] a zo špecifikácie radiča LCD `NV3035C` [15]. Avšak, tieto dokumenty poskytujú iba stručný náhľad na problematiku zobrazovania dát na LCD displeji.

RGB rozhranie sa uplatňuje v konfigurácii, keď sa na module LCD nenachádza integrovaný kontroler obrazovky. Úlohou kontroleru je premostiť iné druhy rozhrania, napr. SPI alebo rozhranie 8080. Pre tieto účely je vybavený aj grafickou pamäťou, v ktorej je uložený kompletný obraz. Typicky je kontroler schopný regulovať aj jas obrazovky, prípadne akcelerovať niektoré základné grafické operácie. Najčastejším typom kontroleru, ktorý sa typicky nachádza na čínskych moduloch je ILI9341 alebo SSD1963.

Nasledovný popis rozhrania bude založený na dvoch režimoch, ktoré podporuje radič `NV3035C` .

Zbernica RGB vo vybranom LCD module je reprezentovaná 24 dátovými vodičmi, ktoré prenášajú informáciu o farbe a podpornými riadiacimi vodičmi. Radič podporuje dve varianty rozhrania RGB. Líšia sa v použití riadiacej signalizácie. Prvý typ, označený ako *HV*[1] mód využíva signály *HSD*, *VSD* a *CLKIN*. Druhý typ, označený ako *DE*[2] mód, využíva *DEN* a *CLKIN*. Oba režimy používajú hodinový signál *CLKIN*. Podľa konfigurácie interných registrov radiča sa pre vzorkovanie môže používať nástupná alebo zostupná hrana. Podľa dokumentácie k LCD modulu predpokladáme, že výrobca displej zapojil v *HV* režime. Usudzujeme to z obrázku č. 3.8, kde si čitateľ môže všimnúť pri symbole *DEN* poznámku v zátvorke „No Connect“. Vychádzame z hypotézy, že tento signál nie je zapojený a preto sa pre prenos a zobrazovanie dát nemôže použiť tzv. *DE* mód.



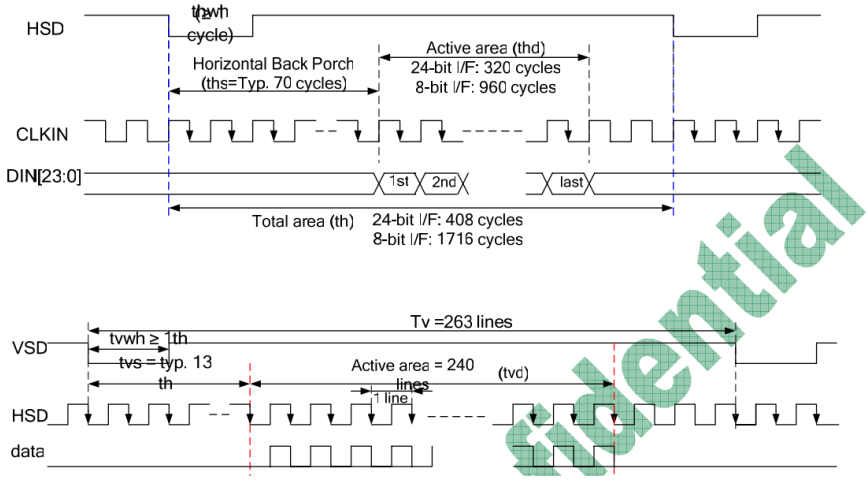
Obr. 3.11: Regióny obrazovky v režime HV Synchronization, zdroj: www.pcper.com

Z historického hľadiska vychádza RGB rozhranie z VGA rozhrania. Pri VGA sa jednotlivé farebné zložky prenášali tromi vyhradenými vodičmi pre každú farbu. Na strane MCU sa nachádzal DAC prevodník, ktorý farebný signál rozdelil na jednotlivé komponenty RGB signálu. Potom sa previedla binárna hodnota farby na analógovú a vystavila sa na vodiče RGB. Na strane monitora sa pomocou týchto analógových veličín ovládalo elektrónové delo, ktoré vystreľovalo prúd elektrónov s intenzitou, ktorá bola priamo úmerná hodnote napätia na vodičoch. S nástupom plochých obrazoviek sa vymenila analógová časť za digitálnu ale princíp zostal rovnaký.

Radič obrazovky vie, kedy má prejsť na ďalší riadok pomocou signálu horizontálnej synchronizácie *HSD*. Pri aktívnej hodnote tohoto signálu dôjde k prepnutiu na ďalší riadok obrazovky. Pri aktívnej hrane hodinového signálu dôjde k prepnutiu na ďalší pixel obrazovky. Pre návrat na počiatočnú pozíciu na celej obrazovke sa využíva signál *VSD*.

[1] Horizontal-Vertical synchronization
[2] Data Enable synchronization

Pri aktívnej hodnote tohoto signálu dochádza k návratu na úplne prvý pixel v oboch osiach X a Y.



Obr. 3.12: Rozhranie RGB, diagram signalizácie [15].

Bohužiaľ, z historických dôvodov je nutné pri zmene riadku alebo pri prechode na novú snímku pridať niekoľko prázdnych zobrazovacích cyklov, tzv „blanking cycles“. V obrazovkách typu CRT sa tieto prázdne cykly používali pre vychýlenie elektrónového dela v prípade posuvu na nový riadok alebo pre návrat na počiatočný bod na obrazovke. Prázdne cykly sa označujú anglickým slovom „porch“. Dĺžka, po ktorú musia byť signály *HSD* a *VSD* v aktívnom stave obyčajne trvá viac hodinových cyklov, počas tejto doby nedochádza k prekresľovaniu obrazu na obrazovke. Celý mechanizmus signalizácie v tomto režime môžeme znázorniť tak, že umelo rozšírime rozlíšenie obrazovky. Skutočná zobrazovaná oblasť sa nachádza v strede, po všetkých stranách sa nachádzajú „porch“ regióny. Obrázok č. 3.11 je dobrým príkladom tejto reprezentácie. V ilustrácii sa používa iné rozlíšenie obrazovky, avšak obrázok je obecný. Jeden hodinový takt v tejto schéme je zhodný so zobrazením jedného pixelu na virtuálnej obrazovke.

Príklad časového diagramu, ktorý zobrazuje signalizáciu na rozhraní RGB sa nachádza na obrázku č. 3.12. Aktívna hrana hodinového signálu je zostupná. Aktívne úrovne synchronizačných pulzov *HSD* a *VSD* je záporná. Radič dovoľuje konfiguráciu aktívnych hrán a úrovní pomocou zápisu do interných registrov.

Tabuľka 3.1: Veľkosti „porch“ regiónov, ktoré sú vypočítané podľa dokumentácie [15]

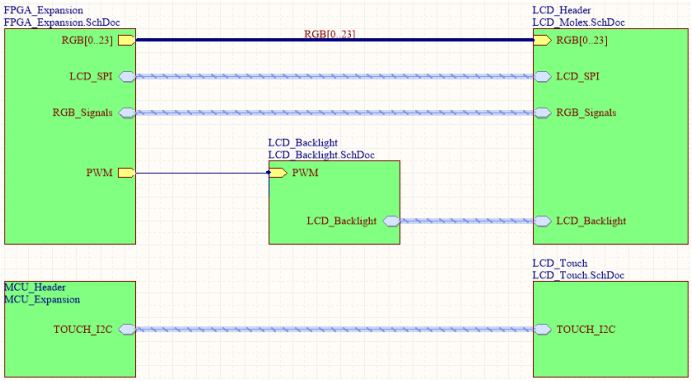
„Porch“ región	Počet taktov
Horizontal Front Porch	70
Horizontal Back Porch	18
Vertical Front Porch	13
Vertical Back Porch	10

Nominálna hodnota hodinového signálu je 6,4 MHz. Po prevode na nanosekundy trvá jedna perióda 156,25 ns. Počet taktov, počas ktorých trvá vykresľovanie jedného riadku je 408. 320 taktov má aktívna časť obrazovky, „porch“ regióny majú dokopy 88 taktov. Z toho vyplýva, že jeden riadok sa vykreslí za $408 \cdot 156,25$ ns $= 63,75$ μs. Celá obrazovka má dokopy 263 riadkov. To znamená, že jedna celá snímka sa vykreslí za $263 \cdot 408 \cdot 156,25$ ns $= 16,76625$ ms. Na základe tejto hodnoty môžeme určiť obnovovaciu frekvenciu na približne 59,64 Hz pri použití hodinového signálu 6,4 MHz.

3.4 Schema a plošný spoj

Návrh elektrickej schémy a dizajn plošného spoja sme vytvorili v programe Altium Designer 17. Vedúci tejto práce nám poskytol zdrojové súbory pre platformu Minerva. Vzhľadom na to, že s návrhom vlastného hardvéru nemáme skúsenosti, sú tieto súbory pre nás referenčný príklad. Podľa osadenia súčiastok na plošnom spoji Minervy vieme osadiť prepojovacie pätice so správnym rozostupom a na presne špecifikované miesta.

V našom návrhu sme sa rozhodli pripojiť obrazovú časť modulu na FPGA a dotykovú časť k mikrokontroleru. Dotykové rozhranie je typu I2C, preto sme vybrali kontakty, ktoré sú v MCU zapojené na internú I2C periférium. Zvyšné dva kontakty sme pripojili na konektory, ktoré nemajú inú alternatívnu funkciu okrem GPIO z dôvodu šetrenia prostriedkov na MCU.



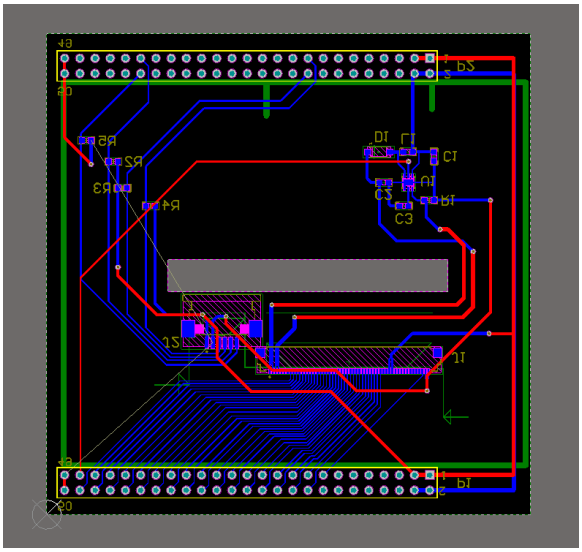
Obr. 3.13: Zapojenie logických komponent v programe Altium Designer.

FPGA čip v nemá dedikované piny a okrem toho musíme zapojiť väčšinu z nich. Preto bolo priradenie ich funkcionality podmienené hlavne jednoduchosťou zapojenia vodičov na plošnom spoji.

Podsvietenie displeja je riadené PWM signálom, ktorý je zapojený na FPGA. Schéma podsvietenia vychádza zo schémy odporúčaného zapojenia z produktovej dokumentácie k LED radiču.

Schéma v programe Altium je hierarchicky rozdelená. Na najvyššej úrovni sa nachádzajú logické komponenty pospájané vodičmi alebo zbernicami. Spodná úroveň definuje vnútorné zapojenie logických komponent. Všetky pasívne súčiastky sú volené v púzdre „0603“ v impe-

riálnej miere. Vďaka dobrej podpore programu Altium sme využili knižnice od spoločnosti Molex a Texas Instruments. Knižnice poskytujú kompletné definície, schématickú značku, footprint aj 3D model súčiastky.



Obr. 3.14: Plošný spoj modulu v programe Altium Designer.

Základná abstraktná reprezentácia elektrického obvodu je na obrázku č. 3.13. Návrh plošného spoja sa nachádza na obrázku č. 3.14. Využíva sa základná dvojvrstvová doska, ktorá má na spodnej strane modré a vrchnej strane červené vodiče. Približne v strede sa nachádza výrez pre prevlečenie ohybného pásika. Molex konektory sú mierne posunuté smerom k stredovému výrezu aby existovala určitá montážna vôľa.

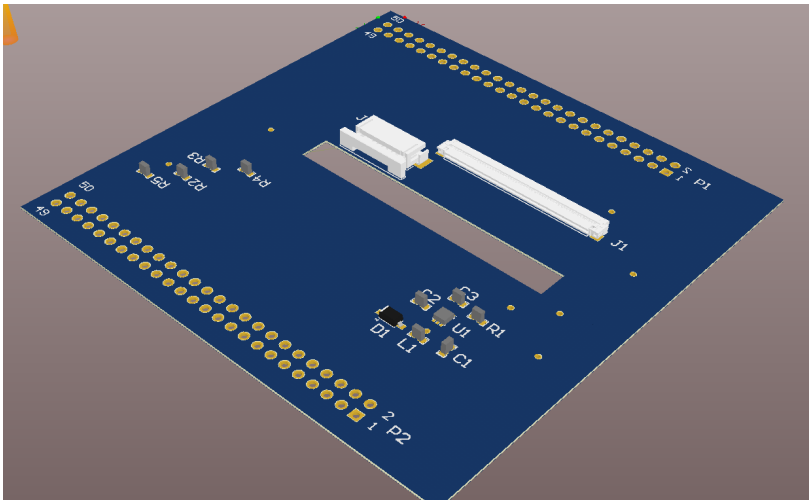
Čierna farba zobrazuje plochu dosky. Zelená vrstva je tzv. mechanická a v programe Altium dáva dizajnérovi možnosť nakresliť si na dosku rôzne anotácie. Vo fyzickej reprezentácii je ignorovaná. Bežne sa pomocou tejto vrstvy vyznačujú hranice súčiastok. Hrubou zelenou čiarou sme nakreslili obrysy LCD modulu a tenšou čiarou sme modelovali ohybný pásik, kvôli presnému umiestneniu Molex konektorov.

Ako bolo spomenuté, využili sme knižnice, ktoré obsahujú aj 3D modely súčiastok. Potom môžeme jednoduchým spôsobom zobraziť 3D model celého plošného spoja, viď obrázok č. 3.15.

Plošný spoj bol v programe Altium exportovaný. Schéma sa nachádza v prílohe. Zdrojové súbory projektu sa nachádzajú na priloženom CD.

3.5 Konštrukcia prototypu

Zdrojové súbory boli odoslané na výrobu do spoločnosti GATEMA v požiadavke na dva kusy. Po doručení bola doska plošného spoja vedúcim práce osadená súčiastkami. Na prototype správne pracuje LED driver spolu s RGB rozhraním. Komunikačná linka 3-Wire

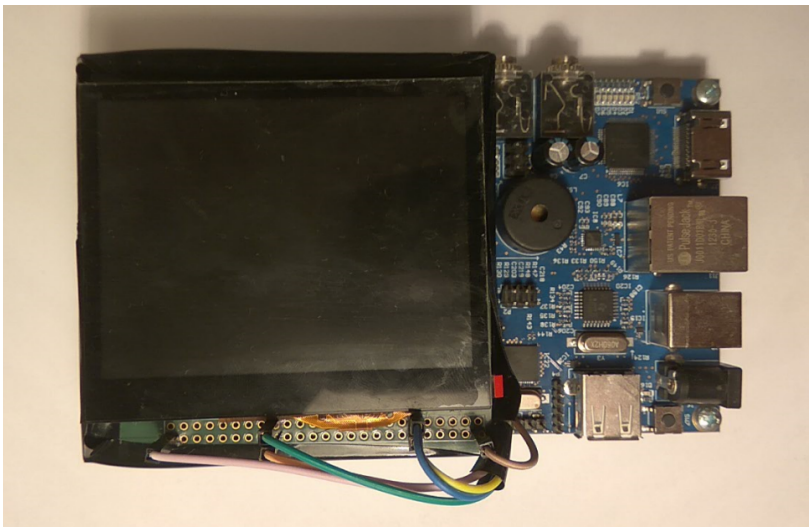


Obr. 3.15: Plošný spoj modulu v 3D zobrazení v programe Altium Designer.

SPI nebola otestovaná. Bohužiaľ nesprávne je navrhnutá pätica pre pripojenie dotykového rozhrania. Číslovanie pinov na konektore začína z opačnej strany.

Aby bolo možné s prototypom pracovať bolo nutné využiť druhú DPS a premostiť pripojenie daných pinov. Po otestovaní sa ukázalo, že toto zapojenie už pracuje správne a už je možné prevádzkovať rozhranie I2C bez problémov.

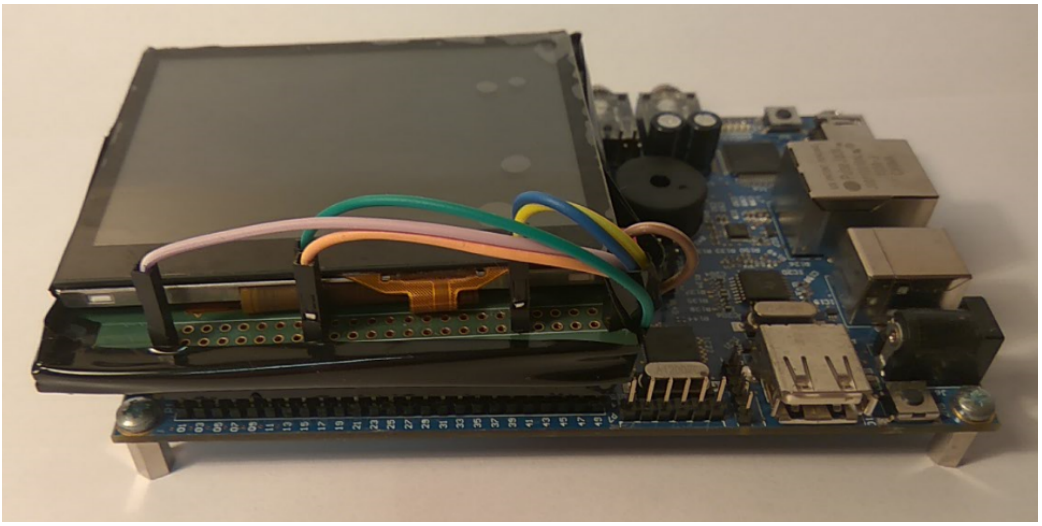
Odhalené konektory boli prekryté izolačnou páskou aby nedošlo k nechcenému skratu. Fixácia displeja je potrebná hlavne kvôli tomu, aby nedošlo k poškodeniu ohybného pásika. Telo displeja nemá na sebe žiadne uchytenie preto je na testovacom prototype prilepené izolačnou páskou po svojom obvode tak, aby sa dal kedykoľvek odlepiť a použiť v inom zapojení.



Obr. 3.16: Prototypová doska osadená displejom pripojená na Minervu.

Ručné osadenie súčiastok je pomerne náročné. LED driver musel byť pájkovaný teplovzdušnou technikou kvoli absencii akýchkoľvek nožičiek na púzdre zdroja. Cievka, ktorá sa nachádza v obvode napájania LED diód musela byť nahradená pretože objednaná súčiastka by sa na pôvodný návrh nezmestila. 54 pinový Molex konektor je rovnako pomerne zložité pripojiť vzhľadom na miniatúrne medzery medzi nožičkami. V tomto smere patrí pánovi Ing. Šimekovi veľká vďaka, že ochotne pomohol pri osadzovaní súčiastok.

Výsledný modul je osadený na expanzné konektory na Minerve. Toto zapojenie je možné vidieť na obrázkoch č. 3.16 a 3.17.



Obr. 3.17: Prototypová doska osadená displejom pripojená na Minervu. Pohľad z boku.

Kapitola 4

Vývojové prostredie a prostriedky pre vývoj

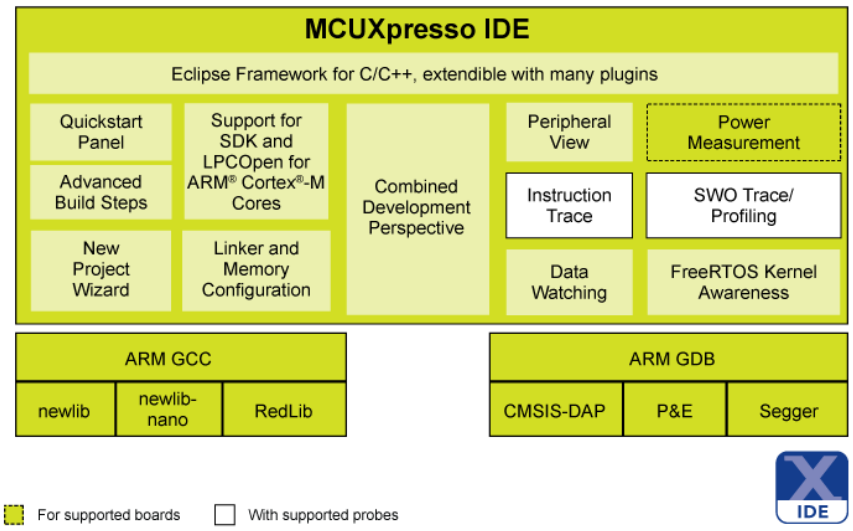
Pre mikroprocesor Kinetis je vytvorené špecializované vývojové prostredie vytvorené priamo na mieru tejto platformy. Nazýva sa KDS – Kinetis Development Studio. Je poskytované zdarma. Pre stiahnutie zo stránok výrobcu je nutná registrácia na webových stránkach. Vývoj zastrešovala spoločnosť Freescale Semiconductor, Inc, ktorá bola neskôr akvizovaná NXP Semiconductors. V októbri roku 2016 bola oznámená spoločnosťou Qualcomm, Inc. správa, že sa chystá v blízkej budúcnosti odkúpiť NXP. Kinetis Development Studio sa v súčasnosti dostalo do verzie 3.2.0.

Sada kompilátorov jazyka C a C++ je založená na projekte GCC v optimalizovanej variante pre ARM. Štúdio je založené na populárnom integrovanom vývojovom prostredí Eclipse. Toto prostredie pridáva výbornú podporu pre parsovanie jazykov C a C++.

Okrem toho ponúka kompletnú podporu pre ladiace nástroje, ktoré sú schopné zobrazovať hodnoty uložené v registroch, ktoré ovládajú periférie. Spolu s tým je v balíku pribalený nástroj Processor Expert. Tento nástroj dovoľuje urýchliť počiatočný vývoj aplikácie tým, že je schopný k jednotlivým pinom asociovať konkrétnu funkciu. Následne sa pomocou jediného tlačidla vygeneruje kód špeciálne pre danú platformu. Processor Expert má v sebe zabudované aj niektoré nízkoúrovňové ovládače periférií. Vzhľadom na to, že sú tieto knižnice napísané pre rôzne rodiny mikrokontrolerov, zdrojový kód ovládačov periférií je komplikovanejší, kvôli podpore ostatných platforiem.

Počas písania tejto práce sa vydalo úplne nové vývojové prostredie MCUXpresso IDE. Tento krát už pod hlavičkou spoločnosti NXP. Rovnako sa na webových stránkach NXP objavili webové nástroje, ktorými je možné nechať si vygenerovať SDK na mieru pre danú platformu. Nástroje ponúkajú aj konfiguráciu pinov a kompletný manažment hodinového signálu. Užívateľ je do značnej miery abstrahovaný od nutnosti manuálne nastavovať všetky inicializačné parametre. Webové nástroje sú schopné vygenerovať potrebný kód automaticky. Zlepšenia, ktoré MCUXpresso IDE prináša spočívajú hlavne v novších sadách kom-

pilačných nástrojov a v novšej verzii Eclipse IDE. Vygenerované SDK obsahuje užitočné nízkoúrovňové ovládače periférií KSDK 2.0.



Obr. 4.1: Blokový diagram vývojového prostredia MCUXpresso, zdroj www.nxp.com.

Pre programovanie FPGA využijeme jazyk VHDL a syntézne nástroje od spoločnosti Xilinx. Hradlové pole Spartan-6, ktoré je osadené na Minerve, už nie je podporované v najnovšej edícii vývojových nástrojov. Posledný kompatibilný balík je ISE Design Suite vo verzii 14.7. Edícia „WebPACK“ je dostupná zadarmo pre vybrané druhy čipov. Pretože vývojové nástroje už nie sú podporované, na operačnom systéme Windows 10 je potrebné urobiť niekoľko drobných zmien, aby bolo možné vývojové prostredie spustiť. Kompatibilita s novými operačnými systémami môže byť dosiahnutá použitím virtuálizácie.

Pre programovanie FPGA od spoločnosti Xilinx je nutný JTAG programovací kábel. K vývoju bol zapožičaný typ „Xilin Platform Cable USB“. Samotné programovanie prebieha v nástroji od spoločnosti Xilinx – „Impact“. Nástroj má minimalistické rozhranie a okrem zápisu do aktívnej konfigurácie FPGA umožňuje zápis aj do konfiguračnej Flash pamäte, ktorá je pripojená k čipu SPI rozhraním. Po spustení sa bootloader na čipe snaží načítať počiatočnú konfiguráciu z Flash pamäte.

Pre správne naprogramovanie je potrebné okrem pripojenia vývojového kábla upraviť programovanie čipu FTDI Vinculum II. Tento čip tvorí rozhranie medzi UART a USB rozhraním. Signály `FDONE`, `FINIT` a `FDONE` nesmú byť ovplyvňované týmto prevodníkom, inak dochádza k chybe programovania v programe Impact. Opatrenia, ktoré je nutné vykonať zahŕňajú priradenie pinov 12, 13 a 14 k periférii `GPIO_A`. Následne na to musia byť tieto piny nastavené ako vstupné. Krátky kus kódu, ktorý je v ukážke č. 4.1 sa musí objaviť v inicializačnej funkcii programu pre Vinculum prevodník.

Užitočný nástroj na analýzu dátovej komunikácie a detekcie chýb je bezpochyby logický analyzátor. Svoje uplatnenie nájde ako pri prvotnom oživovaní modulu, tak aj pri následnom

Ukážka kódu 4.1: Nastavenie vstupných pinov(input) a priradenie ku periférii GPIOA

```
// GPIO_Port_A_1 to pin 12 as Input.
vos_iomux_define_input(12, IOMUX_IN_GPIO_PORT_A_1);
// GPIO_Port_A_2 to pin 13 as Input.
vos_iomux_define_input(13, IOMUX_IN_GPIO_PORT_A_2);
// GPIO_Port_A_3 to pin 14 as Input.
vos_iomux_define_input(14, IOMUX_IN_GPIO_PORT_A_3);
```

testovaní správnej komunikačnej prevádzky periférií na rozširujúcom module. K dispozícii máme 8 kanálový analyzátor kompatibilný s nástrojmi od spoločnosti Saleae.

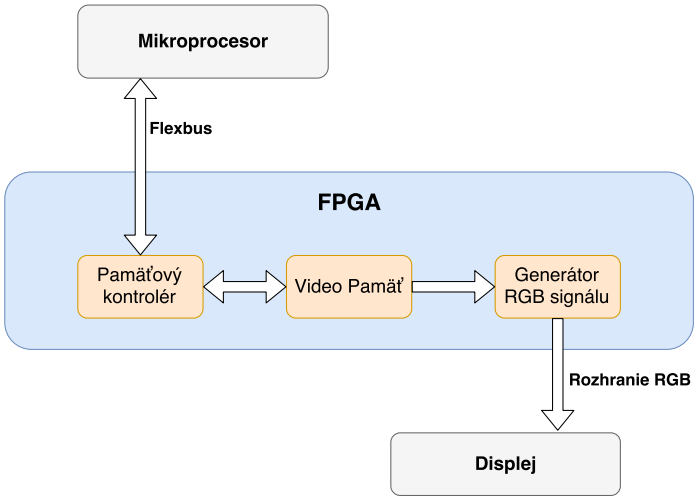
Kapitola 5

Kontroler displeja na FPGA

Ako bolo uvedené v predchádzajúcej kapitole, základom RGB signalizácie je staršie analógové rozhranie VGA. Vzhľadom na to, že musíme neustále obnovovať obraz s konštantnou frekvenciou, je ideálne implementovať kontroler displeja v jednotke FPGA. Podobnosť s VGA rozhraním je v tomto prípade výhodou, pretože pre VGA existujú implementácie vo VHDL, ktorými sa môžeme inšpirovať [20].

Pri návrhu vychádzame z predpokladu, že mikroprocesor musí byť schopný pristupovať do video pamäte. Displej musí byť obnovovaný dostatočne rýchlo a nepretržite. Pri pozastavení vykresľovania displej do približne jednej sekundy stratí obrazovú informáciu a celá obrazovka prechádza do bielej farby.

Základný a zjednodušený model kontroleru, ktorý je implementovaný v FPGA, sa nachádza na obrázku č. 5.1. Jednotlivé funkčné bloky budú popísané samostatne.



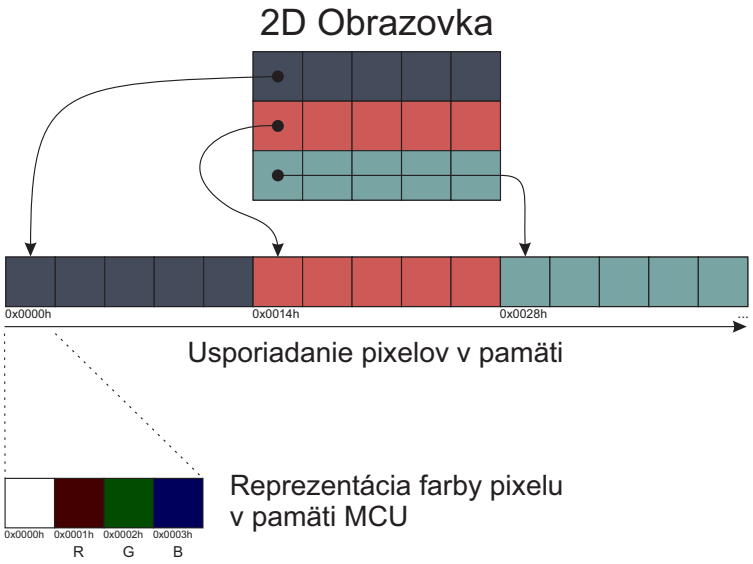
Obr. 5.1: Zjednodušená bloková schéma znázorňujúca hlavné časti nachádzajúce sa na čipe FPGA.

V našom systéme sa video pamäť dostáva do stredu návrhu. Pamäť je zdieľaná medzi generátorom RGB signálu a pamäťovým kontrolerom. Generátor sa stará o riadenie signálov

rozhrania RGB spolu s indexovaním jednotlivých pixelov z obrazovej pamäte. Pamäťový kontroler je exkluzívne využívaný iba mikrokontrolerom pre prístup do obrazovej pamäte. Smer šípok v obrázku znázorňuje, akým smerom prúdia dáta medzi jednotlivými logickými blokmi.

Mikrokontroler je k FPGA pripojený rozhraním FlexBus. Celá schéma komunikácie je popísaná v časti 5.4.

5.1 Grafická pamäť



Obr. 5.2: Ukážka rozprestrenia 2D obrazovy do 1D pamäti. Pole je uložené po riadkoch.

V štádiu návrhu modulu sa predpokladalo, že pre uloženie dát bude použítý DDR2 čip, ktorý sa nachádza na Minerve. Jedna z položiek, ktorá mala byť uložená v tejto pamäti bola grafická pamäť. K tomuto účelu sa na FPGA čipe nachádza hardvérovo implementovaný radič DRAM pamäte typu DDR/DDR2/DDR3, ktorý nezaberá žiadnu kapacitu voľných prostriedkov na čipe, ktoré sú prístupné užívateľovi.

Vlastná implementácia DDR2 radiča je značne obtiažna. Radič musí serializovať viacnásobný prístup do pamäte a riadiť tzv. „refresh“. Všetky pamäte typu DRAM sa musia po určitom čase obnovovať (typicky ~60 ms pre DDR2), inak sa náboj v bunke stratí, v dôsledku čoho sa stráca binárna informácia uložená v bunkách. Vlastná implementácia by zaberala príliš veľkú kapacitu čipu. Extrémne dôležité je zachovať všetky parametre časovania pamäťového čipu. V neposlednom rade by mala implementácia nižšiu rýchlosť prenosu dát a vyšší príkon. Z týchto dôvodov spoločnosť Xilinx pridáva do svojich čipov hardvérový radič. Na internete je dostupných niekoľko projektov, ktoré úspešne riešia čítanie dát z DDR2 pamäte, ktorá je pripojená k FPGA čipu, viď [7].

V tejto práci musí byť pamäť použitá pre uloženie pixelov, ktoré budú neskôr zobrazené na obrazovke. Mapovanie pixelov z obrazovej matice do video pamäti je zobrazené na obrázku č. 5.2. Riadky obrazovej matice sú za sebou uložené v postupnosti od prvého riadku po posledný.

5.2 Natívny kontroler DRAM pamäti

Pokus o komunikáciu natívneho kontroleru s DDR2 čipom sa ukázal byť problematický. Pôvodný návrh sa spoliehal na umiestnenie framebufferu do externého pamäťového modulu. Z tohoto dôvodu musel byť návrh prehodnotený a musel byť prijatý obmedzujúci kompromis. Bohužiaľ sa tieto problémy objavili až po vyrobení prototypového kusu rozširujúceho modulu, takže nebolo možné zmeniť hardvérový návrh. Návrhy na zlepšenie situácie s uložením budú diskutované na konci tejto práce.

Vzhľadom na to, že radič pamäte sa nachádza na čipe, stačí aby bola z prostredia VHDL inštanciovaná šablóna tohoto zariadenia. Pre uľahčenie práce sú pre vývojára dostupné nástroje pre generovanie parametrizovanej šablóny daného typu radiča. Xilinx ISE ponúka pre tento účel Xilinx MIG[1]. Tento nástroj je schopný vytvoriť zdrojové kódy v jazyku VHDL alebo Verilog pre konfiguráciu parametrov, ktorú si zvolí užívateľ.

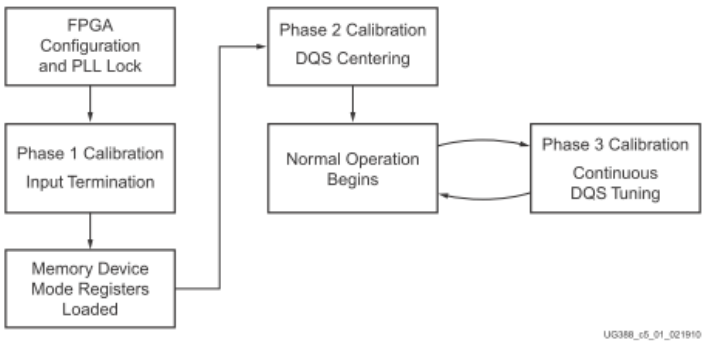


Figure 4-1: **MCB Startup Sequence**

Obr. 5.3: Kalibračná štartovacia sekvencia radiča pamäte na Xilinx Spartan-6 [23].

Jedným z kritických parametrov pre správnu funkciu DDR radiča je časovanie a fyzické usporiadanie buniek v pamäti. V generátore sa pre tento účel nachádza niekoľko predkonfigurovaných pamäťových čipov, ktoré môžu byť zvolené. Pokiaľ sa cieľový čip v generátore nenachádza, ako to je v našom prípade, je užívateľovi umožnené vytvoriť novú komponentu. V nástroji sme zvolili parametricky podobnú pamäť od spoločnosti Micron ako predlohu, a pridali sme náš čip do databázy. Podľa dokumentácie pre pamäťový čip IS43DR16320B sme nastavili správne parametre [8] časovania a frekvencie. Program následne umožní zvoliť si konfiguráciu používateľských portov (obrázok č. 5.4) s ohľadom na bitovú šírku jedného

[1] Memory Interface Generator – nástroj pre generovanie rozhrania k radiču DRAM

slova a smeru dát. Zvolili sme konfiguráciu dvoch 32-bitových portov. Prvý port je určený pre čítanie a zápis dát do video pamäti zo strany mikrokontroleru. Pomocou druhého portu budú dáta iba čítané. Tento port bude pripojený na generátor video signálu.

Po nakonfigurovaní všetkých parametrov sprievodca vygeneruje zdrojové súbory na disk.

Nepríjemné obmedzenie nástroja MIG je, že si sám nevie syntetizovať hodinový signál. Tento signál musí byť syntézovaný používateľom alebo musí byť vytvorený externe mimo FPGA čip. Vstupný kmitočet musí byť rovnaký ako frekvencia zbernice medzi DDR2 čipom a FPGA. Logika radiča túto frekvenciu delí na polovičnú. Touto frekvenciou taktuje svoje vnútorné súčasti, napríklad vyrovnávacie pamäti.

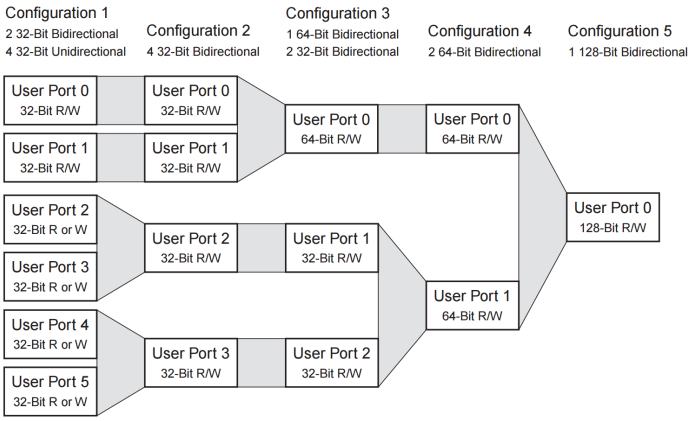
Jediný zdroj hodinového signálu, ktorý je dostupný na FPGA je zdvojený vstup 50 MHz, ktorý je privedený z oscilátoru na Minerve. Týmto signálom je taktované aj jadro mikrokontroleru Kinetis. Najmenšia interná frekvencia, ktorú nástroj MIG umožňuje pre daný pamäťový čip použiť je 125 MHz. Najmenšia frekvencia bola zvolená preto, aby sa zamedzilo chybe časovania pri vyšších taktoch. Vygenerované IP jadro radiča pamäti očakáva na svojom vstupe dvojnásobnú frekvenciu. Zo vstupných 50 MHz sa pomocou PLL komponenty vytvorí signál o frekvencii 250 MHz. Tento signál je privedený na vstup radiča.

Jedným z užitočných vygenerovaných súborov je aj testovací program ktorý inštanciuje DDR radič a vytvorí na danom porte umelú dátovú prevádzku. Tento projekt obsahuje jednoduchý „top layout“, ktorý sa dá použiť pre otestovanie celej pamäťovej cesty. Počas experimentovania s pamäťovým systémom sme používali prevažne tieto súbory.

Po inicializovaní FPGA vykonáva DDR radič kalibráciu zbernice, aby sa zabezpečila stabilná spoľahlivá vysoko-rýchlostná komunikácia. Celý priebeh kalibračnej rutiny je popísaný na stránkach Xilinx, odkiaľ je prebratý obrázok č. 5.3. Úspech je indikovaný logickým signálom, ktorý je pripojený na LED diódou. V momente, keď je kalibrácia hotová (Phase 2, obrázok č. 5.3), sa prechádza na režim normálnej činnosti. Na porty sa umelo generuje dátová prevádzka, aby sa otestoval dátovy prenos. Chyba v tejto fáze je indikovaná signálom, ktorý je pripojený na druhú LED diódu.

Počas testovania sa radič nebol schopný prepúť do fázy normálnej činnosti. Z nezistených príčin sa kalibrácia nepodarila. Určenie časti kalibračnej sekvencie, v ktorej nastala chyba, nie je jednoduché. Časť operácií je implementovaná priamo v hardvéri. Okrem toho sa program v logika v FPGA nedá krokovať tak, ako je známe z programovacích jazykov.

Úspešné nebolo ani riešenie, v ktorom sa kalibrácia preskočila. V tomto stave radič indikoval, že pracuje správne. Avšak, pri pokuse o čítanie z pamäti dochádzalo neustále k chybe. Po vystavení inštrukcie čítania bloku dát musí užívateľský kód čakať, kým sú do vstupného bufferu uložené bajty. Pokiaľ sú dáta do bufferu počas čítania prenesené, zmení sa príznak prázdnosti a užívateľský kód smie čítať dáta z vyrovnávacej pamäti. V našom prípade opakovane dochádzalo k problému, kedy sa po inicializácii čítania vstupné dáta nikdy neobjavili vo vstupnom bufferi. Tento stav spôsobil zablokovanie FlexBus transakcie a mikrokontroler musel byť resetovaný pre obnovenie činnosti.



Obr. 5.4: Dostupné konfigurácie užívateľských portov natívneho pamäťového kontroleru [21].

Možná príčina tohoto problému môže spočívať v tom, že pamäťový čip IS43DR16320B nie je uvedený v zozname kompatibilných a verifikovaných čipov v dokumentácii [21]. Fáza kalibrácie nemusí uspieť z rôznych dôvodov. Podľa manuálu väčšina z nich súvisí so zlým nastavením parametrov v nástroji na generovanie inštančnej šablóny. Prípadne to môže indikovať chybu v hardvéri na strane FPGA alebo pamäťového čipu. Bez dôkladného preskúmania celej pamäťovej cesty nie je možné určiť miesto zdroja chyby.

5.3 Náhradné riešenie

Po zistení, že pamäťový čip DDR2 nebude môcť byť použitý, bolo potrebné zaistiť miesto pre video pamäť. V tejto kapitole sú predstavené možné riešenia daného problému.

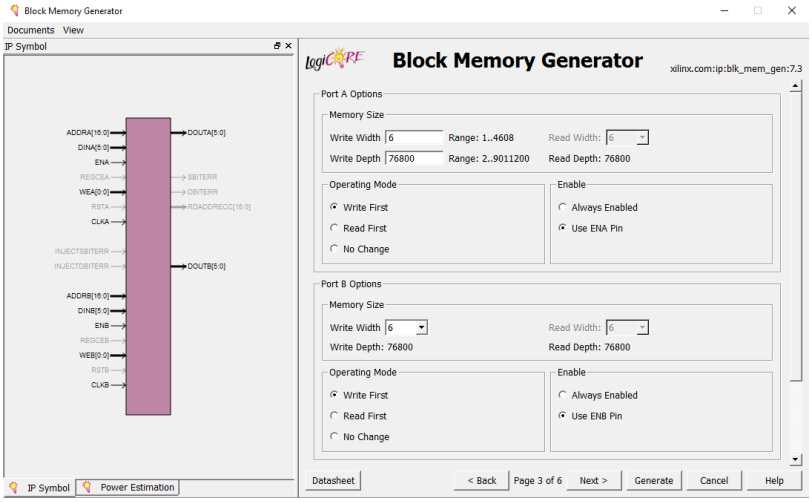
SRAM pamäť mikrokontroleru Kinetis K60 je rozdelená na 2 bloky o veľkosti 64 kB. Problém pri použití tejto pamäte je, že mikroprocesor by nedokázal dostatočne rýchlo obnovovať obrazovku. Jednak by to malo za následok nepríjemné blikanie obrazu alebo neželané obrazové artefakty. Rovnako by to prinieslo veľké nároky na procesorový čas. Dovolíme si konštatovať, že procesor by nemohol vykonávať inú činnosť, ako je obnovovanie obrazu na obrazovke. Nespornou výhodou by bola priestorová lokalita dát (pamäť sa nachádza priamo na mikrokontroleri) a extrémne malá prístupová doba. Alokácia framebufferu na MCU by znamenala zníženie pamäti RAM, ktorá môže byť použítá pre program bežiaci na MCU. Z hľadiska kapacity sa do pamäti nezmestí celá obrazová pamäť pri konfigurácii 8 bitov na jednu zložku farby. Farebný priestor by musel byť redukovaný minimálne dvakrát. Pre uloženie kompletného obrazu v hĺbke 24 bitov je potrebné 225 kB pamäti. Dostupnej je však iba kapacita 128 kB.

Druhým riešením, ktoré sa naskytá, je použitie externého pamäťového čipu. Tento čip by musel byť umiestnený na DPS rozširujúceho modulu. Štandardným riešením je použitie SRAM pamäte s dostatočnou kapacitou. Typické IO rozhranie SRAM čipu sú adresová a dátová zbernica plus niekoľko prídavných vodičov. Pre pamäť s veľkosťou 4 Mib je šírka

dátovej časti zbernice 16 bitov a adresová časť má 18 bitov. Bohužiaľ, počet voľných kontaktov na expanznom FPGA rozhraní je iba 11. Z tohoto dôvodu sa nedá použiť ani varianta oddelenia video pamäti do externého SRAM čipu.

Tretím a posledným možným riešením je využitie integrovaných pamäťových blokov na FPGA čipe. Produktové označenie Spartan-6 čipu je XC6SLX9. Podľa materiálov k celej rodine FPGA čipov [22] sa na mikročipe nachádza celková bloková pamäť o veľkosti maximálne 576 kb. Táto pamäť je organizovaná v 32 bunkách, každá o kapacite 18 Kkb. Okrem blokovej pamäti sa dá využiť aj tzv. distribuovaná pamäť, ktorá je vytvorená z CLB[2] blokov. Jej maximálna kapacita je 90 kb. Nevýhodou je, že pri využívaní CLB pre ukladanie dát sa zaberá voľné miesto pre užívateľskú logiku.

Vývojové nástroje od Xilinx obsahujú generátor blokovej pamäte. V sprievodcovi sme vytvorili dvojportovú pamäť s parametrami podľa obrázku č. 5.5.



Obr. 5.5: Nastavenie parametrov v generátore blokovej pamäti.

Kapacita blokovej pamäte je príliš malá, aby sa do nej dala uložiť kompletná video pamäť. Kvôli tomu je nutné významne redukovať bitovú šírku jedného slova v pamäti. Modul LCD displeja je schopný zobraziť 256 rôznych intenzít pre každú farebnú komponentu. Dokopy zaberajú 3 farebné zložky až 24 bitov. Pre využitie blokovej pamäte musíme redukovať bitovú hĺbku každej farby z pôvodných 8 bitov na 2. Počet zobraziteľných farieb sa zníži z 2^{24} na 2^6. Toto výrazné obmedzenie bude diskutované v závere práce.

Z hľadiska časovania je umiestnenie framebufferu do blokovej RAM krok, ktorý zníži zložitosť radiča RGB signalizácie. Prístupová doba do SRAM pamäte je malá. Po vystavení adresy na port trvá iba jeden hodinový takt kým sa dáta objavia na výstupnej zbernici. Každý port má svoj vlastný nezávislý hodinový vstup, čo umožňuje kompletne oddeliť rozhranie pre zápis do pamäte, ktorým mikroprocesor aktualizuje dáta vo video pamäti,

[2]CLB – Configurable Logic Block, základná jednotka pre užívateľskú logiku

a rozhranie pre čítanie dát z pamäte, pomocou ktorého bude radič RGB rozhrania generovať obrazový signál.

5.4 Komunikačné rozhranie

Mikrokontroler je k FPGA čipu pripojený pomocou dedikovanej zbernice FlexBus. Táto zbernica je riadená perifériou na mikročipe a je mapovaná do adresového priestoru mikrokontroleru. Najväčšia šírka slova, ktorú zbernica podporuje je 32 bitov, teda 4 bajty. Fyzicky je tvorená sadou riadiacich signálov a 32 vodičmi, ktoré slúžia pre prenos adresy aj dát v oboch smeroch.

Komunikáciu na zbernici vždy iniciuje mikroprocesor (Master) a klientské zariadenie odpovedá (Slave). Avšak, ak je to vyžadované, jeden z riadiacich signálov `FXB_TA` (obrázok č. 5.6) môže byť použitý ako indikácia, že došlo k udalosti na klientskom zariadení, ktoré má Master kontroler spracovať.

Zbernica môže byť nakonfigurovaná v nasledovných režimoch:

- Multiplexovaná 32-bitová adresa a 32-bitové dáta
- Multiplexovaná 32-bitová adresa a 16-bitové dáta
- Multiplexovaná 32-bitová adresa a 8-bitové dáta
- Dedikovaná 32-bitová adresa a dedikované 32-bitové dáta

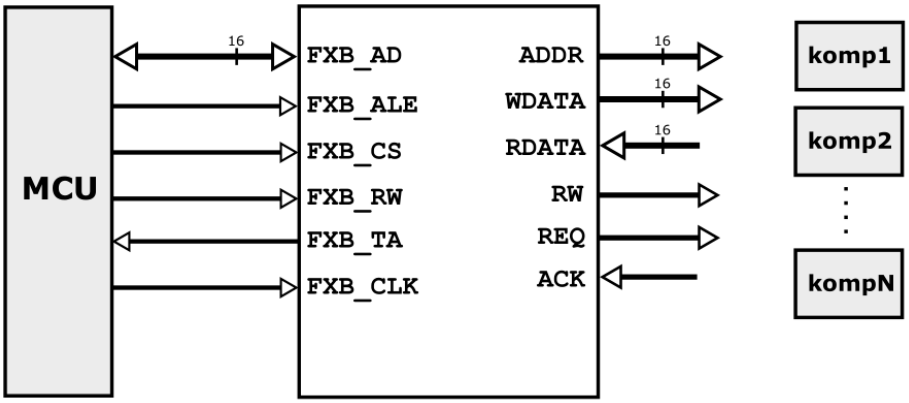
Z hľadiska časovania signálov na zbernici je v prvom takte vždy prenesená najprv adresa a po nej nasledujú dáta. Napriek tomu, že Flexbus podporuje aj režim, kedy je adresová a dátová časť zbernice fyzicky oddelená, zapojenie na Minerve tento režim neumožňuje.

Vrámci svojej diplomovej práce sa pán Buchta zaoberal sprevádzkovaním rozhrania medzi mikrokontrolerom a FPGA čipom [1]. V svojom riešení vytvoril radič zbernice v jazyku VHDL, na ktorý je možné pripojiť ľubovoľne veľa komponentov. Jeho implementácia využíva režim 16-bitových multiplexovaných dát s adresnou časť zbernice. V tento práci sme zbernicu rozšírili na 32 bitov, aby bol prenos dát efektívnejší.

Pán Buchta navrhol architektúru, v ktorej existuje jeden hlavný radič zbernice FlexBus. Jeho úlohou je abstrahovať ostatné zariadenia od zložitosti riadenia zbernice. Všetky entity, ktoré potrebujú pristupovať k zbernici, musia byť k tomuto radiču pripojené. Pamäťový kontroler, ktorý sa nachádza na obrázku č. 5.1, je zložený z radiča FlexBus od pána Buchty a jednoduchého prevodníka, ktorý číta a zapisuje do zdieľanej pamäti.

V štandardnom nastavení je operácia na FlexBus rozhraní blokujúca. Mikrokontroler čaká, kým sa na zbernici vystaví signál indikujúci, že transakcia je dokončená. Následne na to pokračuje v normálnej činnosti.

Prevodník, ktorý dekóduje požiadavky na čítanie a zápis do video pamäti je implementovaný ako jednoduchý stavový automat. Aktivácia komponenty prebieha, pokiaľ je

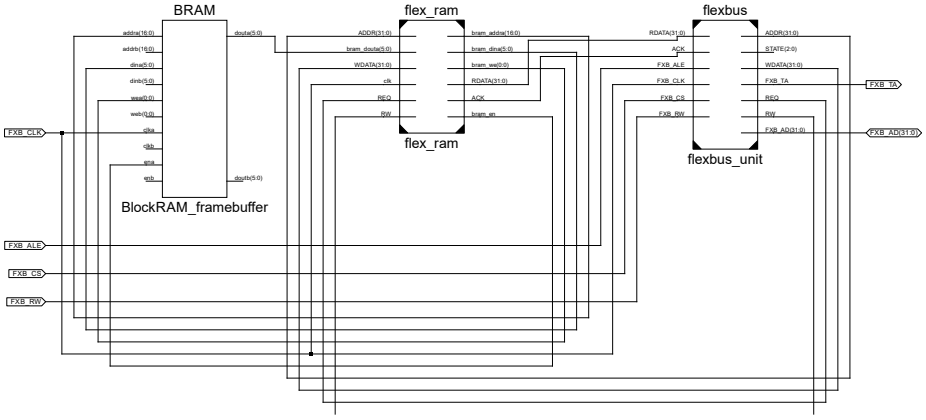


Obr. 5.6: Zapojenie užívateľských komponent na FlexBus kontroleri a pripojenie kontroleru k MCU [1].

na FlexBus vystavená správna adresa. Následne na to sa vykoná čítanie alebo zápis do video pamäti.

Čítanie trvá dva hodinové cykly. V prvom cykle je na adresnú časť pamäťového portu vystavená adresa. V nasledujúcej hrane hodinového cyklu sú dáta z pamäti zapísané na zápisovú zbernicu FlexBus radiča `RDATA`.

Zápis dát trvá jeden hodinový cyklus. Adresa `ADDR`, dáta `WDATA` aj povoľovací signál sú vystavené na pamäťovú zbernicu naraz. Kratší zápisový čas je výhodou, pretože sa predpokladá, že do grafickej pamäte sa bude prevažne zapisovať. Data v pamäti sú reálne modifikované až pri nástupnej hrane druhého cyklu.



Obr. 5.7: Blokový diagram zapojenia pamäťového kontroleru pre prístup do video pamäte.

Zbernica FlexBus je mapovaná do pamäte mikrokontroleru. Rozsah sa začína na adrese 0x60000000h a končí na adrese 0x7FFFFFFFh. Pokiaľ počas behu programu dôjde k adresácii tejto pamäti, potom periféria inicializuje transakciu.

Komponenty, ktoré sú pripojené na FPGA Flexbus radič, sú zodpovedné za správne rozdelenie pamäťového priestoru. Každá komponenta musí dekódovať adresnú časť zber-

Tabuľka 5.1: Redukcia 32-bitových dát na 6-bitovú farbu pomocou výberu najnižších bitov.

Unused [31:24]	Red [23:16]	Green [15:8]	Blue [7:0]
xxxxxxxx	xxxxxxRR	xxxxxxGG	xxxxxxBB

nice `ADDR`, ktorá sa nachádza na obrázku č. 5.6. Začiatok novej transakcie je indikovaný vystavením „1“ na vodič `REQ`. Pokiaľ adresa spadá do aktívnej oblasti danej komponenty, potom je táto komponenta povinná transakciu spracovať a po dokončení nastaviť signál `ACK` na hodnotu „1“. Ak by k tomu nedošlo, mikrokontroler sa zastaví v nekonečnom čakaní, pretože transakcia sa nikdy nedokončí.

Interná FlexBus zbernica je zdieľaná, takže v jedom momente môže byť aktívne maximálne jedno zariadenie. Ostatné komponenty, ktoré nie sú práve adresované, musia mať svoje výstupy v stave vysokej impedancie. Tento stav je vo VHDL označovaný písmenom „Z“.

Video pamäť sa nachádza na počiatku pamäťového rozsahu FlexBus 0x60000000h. Každý pixel je z pohľadu MCU reprezentovaný 32 bitmi. Druhý pixel sa nachádza na adrese 0x60000004h. Takto je mapovaných všetkých 76800 obrazových bodov.

Je dôležité upozorniť na to, že z pohľadu MCU je pixel 32-bitová premenná, zatiaľ čo z pohľadu video pamäti je pixel iba 6-bitová premenná. Ako bolo spomenuté, súvisí to s obmedzenou kapacitou blokovej RAM pamäte.

Pri prenose 32-bitového pixelu je každá farebná zložka modelu RGB reprezentovaná jedným bajtom. Posledný (najvyšší) bajt nie je momentálne použitý. Postup redukovania 24-bitového farebného priestoru na 6-bitový je jednoduchý. Výberú sa 2 najnižšie bity pre každú farebnú zložku, tak ako je zobrazené v tabuľke č. 5.1.

Nevyužitý bajt môže byť v budúcnosti použitý na kódovanie priehľadnosti. Farebný model by potom bol charakterizovaný schémou ARGB, kde písmeno „A“ značí *Alpha* kanál.

Spätná konverzia 6-bitovej farby na 32-bitové dáta, ktoré budú prenesené po FlexBus zbernici, sa vykoná tak, že pre každú farebnú zložku sa doplnia zľava bity s hodnotou „0“. Najvyšší bajt ktorý sa nepoužíva je konštantne nastavený na nulovú hodnotu.

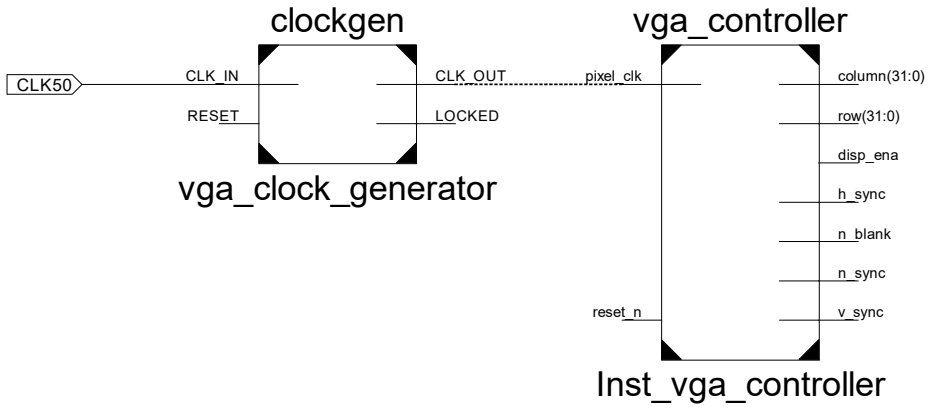
5.5 Generovanie signalizácie RGB

Najdôležitejšia úloha pre proces zobrazovanie dát na displeji je správne generovanie riadiacej signalizácie RGB rozhrania. Problém s ktorým je potrebné sa vyrovnať je presnosť časovania signálov horizontálnej a vertikálnej synchronizácie. Dôležité je presne zachovať „porch“ regióny. Detailný popis rozhrania RGB sa nachádza v sekcii č. 3.3.2.

Bloková schéma zdroja RGB signálu sa nachádza na obrázku č. 5.8. Základom pre generovanie signalizácie je syntéza hodinového signálu o frekvencii 6,4 MHz. Na obrázku sa tento signál nazýva `CLK_OUT`, prípadne `pixel_clk`. Počiatočná vstupná frekvencia z oscilátoru je

50 MHz, označenie `CLK50`. Tento signál vstupuje do zabudovanej PLL komponenty, ktorá je inštanciovaná pod menom `vga_clock_generator`.

Vývojové nástroje Xilinx poskytujú nástroj pre konfigurovanie syntézy hodinového signálu „Clocking Wizard“.



Obr. 5.8: Bloková schéma generátora RGB signalizácie spolu s generátorom hodinového signálu.

Signál o obnovovacej frekvencii 6,4 MHz vstupuje do bloku `Inst_vga_controller`. Táto komponenta je zodpovedná za správne generovanie synchronizačných signálov `h_sync` a `v_sync`, ktorých sémantika a je popísaná v kapitole 3.3.2. Vnútorná implementácia je prebraná zo stránky [9]. Pre použitie na našom LCD module stačí zmeniť definície „porch“ regiónov. Aktuálne nastavenie je možné vidieť v ukážke č. 5.1.

Ukážka kódu 5.1: Caption example.

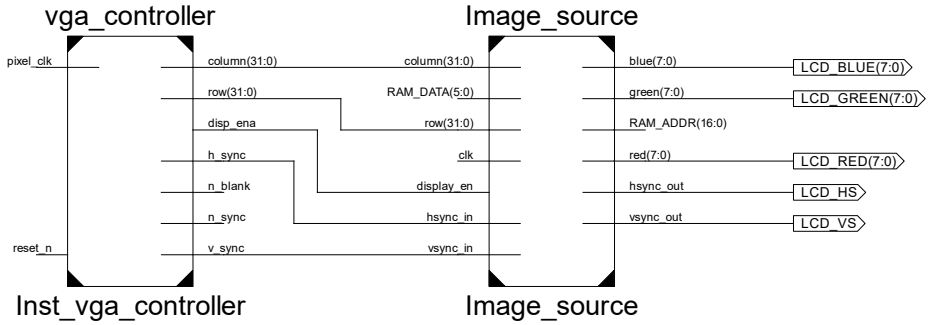
```
h_pulse : INTEGER := 1; --horiztonal sync pulse width in pixels
h_bp : INTEGER := 68; --horiztonal back porch width in pixels
h_pixels : INTEGER := 320; --horiztonal display width in pixels
h_fp : INTEGER := 18; --horiztonal front porch width in pixels
h_pol : STD_LOGIC := '0'; --horizontal sync pulse polarity (1 = positive, 0 = negative)
v_pulse : INTEGER := 1; --vertical sync pulse width in rows
v_bp : INTEGER := 13; --vertical back porch width in rows
v_pixels : INTEGER := 240; --vertical display width in rows
v_fp : INTEGER := 10; --vertical front porch width in rows
v_pol : STD_LOGIC := '0'); --vertical sync pulse polarity (1 = positive, 0 = negative)
```

Okrem generovania synchronizačných signálov sa v bloku `Inst_vga_controller` nachádza počítadlo riadkov a stĺpcov. Výstupy sú použité pre indexovanie pixelovej pamäti. Blok s názvom `Image_source` je pripojený na výstup generátoru signálov. Jeho úlohou je v správnom časovom okamihu vystaviť na zbernicu správne obrazové dáta. Výhodou využitia tejto abstrakcie je skutočnosť, že prístup do pamäte môže obecne trvať stovky hodinových cyklov. Použitím bloku `Image_source` je možné dáta načítať dopredu a uložiť ich do vyrovnávacích

pamätí. Tento princíp sa používa pri načítavaní pixelov z video pamäti uloženej v DRAM čipe, podľa zdroja [7]. V našom prípade trvá prístup do pamäti vždy jeden hodinový cyklus. V prvom takte sa na vstupnom port vystaví adresa. V nasledujúcom takte je na výstupe hodnota adresovanej pamäťovej bunky.

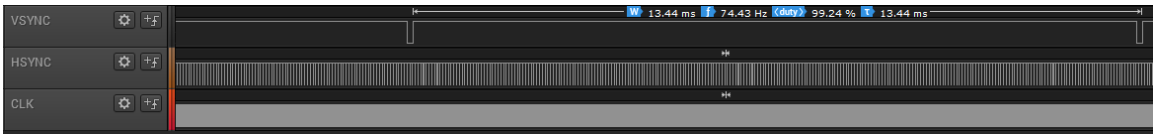
Implementácia `Image_source` je do značnej miery jednoduchšia oproti prípadu, kedy by sa používal DRAM čip. Vzhľadom na konštantné oneskorenie prístupu do SRAM pamäti stačí o jeden hodinový cyklus pozdržať signály `h_sync` a `v_sync`. Dochádza k zreťazenému spracovaniu, kedy je adresa vystavená na pamäťový port vždy o jeden cyklus skôr, ako má byť pixel na obrazovke zobrazený.

Pre prípad s DRAM pamäťou je teoreticky potrebné upraviť iba túto komponentu. Ideálnym riešením je načítať dáta do bufferu počas zobrazovania oblasti „horizontal back porch“. Počas vykresľovania pixelov na obrazovku hrozí podtečenie pamäti, keď sa vyčerpajú predčítané položky v bufferi. Preto je potrebné dostatočne skoro iniciovať nové čítanie z pamäti aby tento závažný stav nenastal. Prejavom problému podtečenia pamäti by bol stav, kedy by sa na obrazovke v jednom riadku objavovali rôzne dlhé tmavé medzery medzi za sebou idúcimi pixelmi.



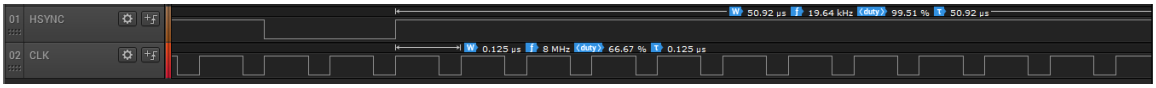
Obr. 5.9: Bloková schéma zapojenia generátora RGB signalizácie a bloku, ktorý generuje obrazové dáta.

Signalizácia, ktorá vstupuje do modulu LCD bola pre účely testovania zaznamenaná logickým analyzátorom od spoločnosti Saleae. Overili sme, že riadiace signály `h_sync` a `v_sync` sú generované v správnom čase a so správnou periódou. Pri pokusnom meraní sme navýšili taktovaciu frekvenciu hodinového signálu na 8 MHz. Modul bol aj naďalej schopný pracovať bez akýchkoľvek viditeľných obrazových artefaktov. Na obrázku č. 5.10 sa nachádza záznam z merania.



Obr. 5.10: Signalizácia RGB generovaná na FGPA, ktorá je zachytená logickým analýzatorom.

Na obrázku č. 5.10 je možné vidieť signály `h_sync` a `pixel_clk`. Nad signálmi je vyznačená perióda a frekvencia signálu. Zaujímavým zistením je, že hodinový signál nemá pracovný cyklus 50. % Približne 66 % času je v stave HIGH.



Obr. 5.11: Priblížené signály horizontálnej synchronizácie (hore) a hodinový signál (dole).

5.6 Obmedzenia aktuálneho riešenia

Skutočnosť, že v tejto práci sa nepodarilo využiť DDR2 pamäť, ktorá je integrovaná priamo na Minerve, kladie veľké obmedzenia na možnosti zobrazovania farieb na displeji.

V prvom rade je to počet zobraziteľných farieb. Počet dostupných farieb je 64 (2^6). Aplikácie budú musieť využívať obmedzenú paletu farieb. Nebude možné vytvárať farebné prechody. Obmedzená alebo chýbajúca bude podpora efektu priehľadnosti.

Tabuľka 5.2: Prevodná tabuľka intenzity vstupnej farebnej zložky na výstupnú.

Hodnota vstupnej farebnej zložky	0	1	2	3
Hodnota výstupnej farebnej zložky	0	86	172	255

Hodnota intenzity jednej farebnej zložky v rozsahu 0 – 3 (2 bity) sa pomocou prevodnej tabuľky č. 5.2 prevádza na intenzitu v rozsahu 0 – 255. Prevod je zostavený tak, aby sa súmerne pokryl celý výstupný interval iba 4 úrovňami.

Nemožnosť ukladať dáta do DRAM čipu je obmedzujúcim faktorom pre GUI knižnicu, ktorá bude popísaná v nasledujúcej kapitole. V budúcnosti je možné pridať aditívnu logiku priamo do FPGA čipu, ktorá by vykonávala časovo náročné operácie, napríklad prekreslenie obrazovky zvolenou farbou. Mixovanie farieb pri použití priehľadnosti, farebné prechody. Pokiaľ bude na displeji zobrazených niekoľko okien, môže logika na FPGA automaticky riešiť viditeľnosť a prekrývanie okien. K tomuto účelu je potrebné mať k dispozícii kompletný grafický obraz každého viditeľného okna. Okrem iného je možné zostaviť nízkoúrovňový rasterizér grafických primitív priamo na FPGA a docieliť tým hardvérovú akceleráciu vykresľovania.

Jediná výhoda aktuálneho riešenia spočíva v krátkej prístupovej dobe do video pamäti.

Rozširujúci modul displeja obsahuje radič LED podsvietenia. Úroveň jasu je možné ovládať PWM signálom. Vrámci testovania hardvéru sme overili, že ovládanie pracuje správne. V našej implementácii sa PWM generátor nenachádza. Aby mohol mikrokontroler ovládať jas obrazovky je potrebné pridať novú komponentu na FlexBus radič, ktorá by zabezpečila generovanie signálu s potrebnou šírkou.

Na LCD module sa nachádza kontroler NV3035C. Pomocou rozhrania 3-Wire SPI je možné nastavovať niektoré parametre RGB protokolu a takisto regulátorov napätia pre ovládanie TFT tranzistorov. Implementácia tohoto protokolu však nie je kritická pre zobrazovanie obrazu na obrazovke. Okrem toho, parametre stačí nastaviť iba raz po resetovaní radiča. Potom je aktívna komunikácia s radičom zbytočná. Z tohoto dôvodu sme implementáciu vynechali ale pripravili sme správne mapovanie portov v `UCF`[3] súbore.

5.7 Použité prostriedky a zdroje

Nástroj Xilinx ISE poskytuje po každej kompilácii zdrojových súborov detailný rozbor obsadenia prostriedkov na FPGA čipe.

Implementácia radiča displeja a pamäťového kontroleru na FPGA čipe je veľmi efektívna z pohľadu nárokov na zdroje. Z dostupnej celkovej kapacity „Slice“ registrov je použitých približne 1 %. Z 32 blokov 18kb pamäťových buniek je zabratých 27. Tvorí to 84 % celkovej kapacity blokovej pamäte. Je to daň za alokáciu video pamäti priamo na FPGA čipe. Tabuľka č. 5.3 zachytáva použitie dôležitých prostriedkov.

Pre vytvorenie hodinového signálu o správnej frekvencii sme využili 1 z dvoch jednotiek PLL_ADV. V prípade potreby sa dá využiť existujúca jednotka pre vytvorenie ďalších hodinových výstupov o inej frekvencii. Vývojár musí počítať s tým, že jednotka je už nakonfigurovaná pre optimálny výstup 6,4 MHz a ostatné výstupné porty nemôžu generovať ľubovoľné frekvencie.

Tabuľka 5.3: Použité zdroje na FPGA.

Názov zdroja	Počet využitých prostriedkov	Celkový počet prostriedkov	[%]
Slice Registers	120	11,440	1 %
Slice LUTs	102	5,720	1 %
IO ports	79	200	39 %
RAMB16BWERs	27	32	84 %
BUFG/BUFGMUXs	3	16	18 %
PLL_ADVs	1	2	50 %

Počas prevádzky je možné si povšimnúť, že FPGA čip je horúci. Presná teplota čipu nie je známa ale pri dotyku prstom čip páli. Vysoká teplota nie je vhodná pre generátor hodinového signálu, pretože výstupný signál môže mať posunutú frekvenciu. Okrem toho môžu trpieť iné súčasti čipu, pretože vysoká teplota znižuje životnosť súčiastok.

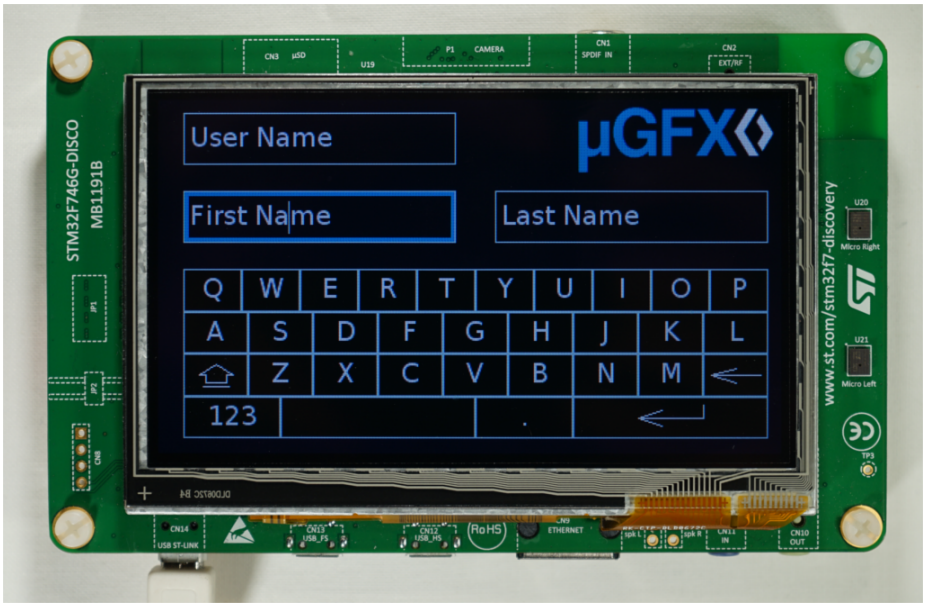
[3]Súbor, ktorý popisuje mapovanie logických portov na fyzické porty na puzdre čipu

Kapitola 6

GUI knižnica

Druhou časťou tejto práce je vytvorenie knižnice pre grafické užívateľské rozhranie. Návrh knižnice musí umožňovať užívateľovi jednoducho pridať na obrazovku prvky užívateľského rozhrania. Pre vývoj používame nové IDE prostredie MCUXpresso od spoločnosti NXP.

Pri vývoji robustnej knižnice na mikrokontroler je treba brať do úvahy obmedzené výpočetné zdroje. Vykresľovanie nesmie trvať príliš dlho, aby sa neblokovali iné procesy, ktoré mikrokontroler riadi. Rovnako tak treba šetrne zaobchádzať s operačnou pamäťou a zásobníkom. Mikrokontroler, ktorý je na Minerve má k dispozícii až 128 kB SRAM pamäti. Populárny mikrokontroler Arduino Uno s čipom Atmega 328P disponuje iba 2 kB operačnou pamäťou. Aj napriek malej pamäti je možné v obmedzenej miere pripojiť k Arduinu displej a vykresľovať naň jednoduché GUI prvky. Veľkosť Flash pamäte pre kód je v prípade Kinetisu K60 až 512 kB. Tento priestor je viac ako dostatočný pre bežné úkony. Rovnako tak takt mikroprocesoru môže byť až 100 MHz.



Obr. 6.1: Ukážka schopností grafickej knižnice μGFX na dotykovom TFT paneli [6].

Z implementačného hľadiska sme sa rozhodli pre využitie jazyka C++. Elementy, ktoré sú vykreslené sa abstrahujú do objektov. Pre tento jazyk sme sa rozhodli aj napriek tomu, že v efektívnosti nemusí prekonať aplikáciu napísanú v C. Domnievame sa, že knižnica bude jednoduchšia na údržbu a užívateľsky prívetivejšia.

Existujúce riešenia, ktoré sú na trhu dostupné, tvoria dve skupiny. Do prvej patria knižnice, ktoré sú vyvíjané jednotlivcami. Najrozšírenejším zástupcom tejto skupiny je knižnica s názvom μGUI [10]. Druhú skupinu tvoria komerčne dostupné knižnice. Zástupca tejto skupiny je μGFX [6]. Spoločnosť ponúka svoju knižnicu na osobné nekomerčné použitie zdarma. Ku knižnici sú dokonca dostupné aj WYSIWYG nástroje pre tvorbu GUI rozhrania. Táto knižnica je zo všetkých dostupných najkomplexnejšia, čo sa týka samotného kódu, podporovaného hardvéru, doplnkových nástrojov a veľkej komunity priaznivcov.

Väčšina knižníc je napísaná v jazyku C, avšak je bežné v nich nájsť formu dedenia pomocou štruktúr v jazyku C. Z objektového programovania si vývojári osvojili formu polymorfizmu v jazyku C, kedy vo vlastnej réžii spravujú tabuľku virtuálnych funkcií. Dôvod, prečo sú knižnice napísané v C môže byť vlastná správa pamäti alebo podpora pre hardvér, na ktorý neexistuje C++ kompilátor.

Našu knižnicu experimentálne tvoríme v jazyku C++. Pokúsime sa využiť natívny spôsob dedenia a polymorfizmu v objektovom jazyku. Pri využití štandardných STL kontajnerov v C++ a STD knižnice je predpoklad, že knižnica bude užívateľsky prívetivá. Rýchlosť knižnice nemusí byť nižšia oproti jazyku C. Avšak, zvýši sa veľkosť výsledného kódu a nároky na pamäť.

Celá knižnica je vnorená do menného priestoru `GUI`. V nasledujúcom výklade sa nebudeme odkazovať na kompletnú cestu k triede napr. `GUI::Button`. Namiesto tohu budeme písať iba `Button`.

6.1 NXP SDK

Spoločnosť NXP pre rodinu mikrokontrolerov Kinetis K60 pripravila sadu ovládačov tzv. MCUXpresso Software Development Kit. K dispozícii je webový konfigurátor, v ktorom sa dajú do základného balíka pridať voliteľné knižnice. V SDK sa nachádzajú štandardné hlavičkové súbory pre konfigurovanie ARM cortex M jadier CMSIS. Okrem toho je súčasťou základného balíka Kinetis SDK (KSDK). Tento Kit obsahuje nízkoúrovňové ovládače pre periférie, ktoré sú na mikrokontroleri dostupné. Poskytujú základné funkcie pre abstrakciu od hardvéru.

Ovládače sú dobre popísané a je k nim dostupný podrobný manuál s typickým príkladom použitia. Po založení nového projektu sú automaticky vytvorené súbory, ktoré inicializujú mikrokontroler. Súbor `pin_mux.{h,c}` priraduje konkrétnu funkciu pinom a zapína hodiny na použitých periférіach. V `clock_config.{h,c}` je umiestnený manažment taktovacej frekvencie. V poslednom súbore `board_.{h,c}` sa nachádza hlavne inicializácia Debug rozhrania.

Pre ladiaci výstup sa používa periféria `UART5`. Na počítači stačí otvoriť sériový port s rýchlosťou 115200 baudov za sekundu.

Ukážka kódu 6.1: Zápis na ladiaci výstup

```
#include <fsl_debug_console.h>
...
DbgConsole_Printf("Hello World\r\n");
```

V knižnici sú využívané prevažne funkcie pre riadenie I2C rozhrania a FlexBus rozhrania. Všetky potrebné súbory z SDK balíka musia byť distribuované spolu s GUI knižnicou aby bola zaistená kompatibilita.

6.2 Architektúra

Grafické užívateľské prostredia nás sprevádzajú už od polovice 70. rokov minulého storočia. Odvtedy prešli rozsiahlym vývojom až sa ustálili v konceptoch, ktoré používame v súčasnosti. Rýchly vývoj mobilného segmentu a IOT v poslednej dekáde podnietil vzostup nového druhu ovládania zariadení. Zatiaľ čo v desktopovej sfére drží pevne svoju pozíciu klávesnica a myš, v mobilnej sfére sú to dotykové obrazovky.

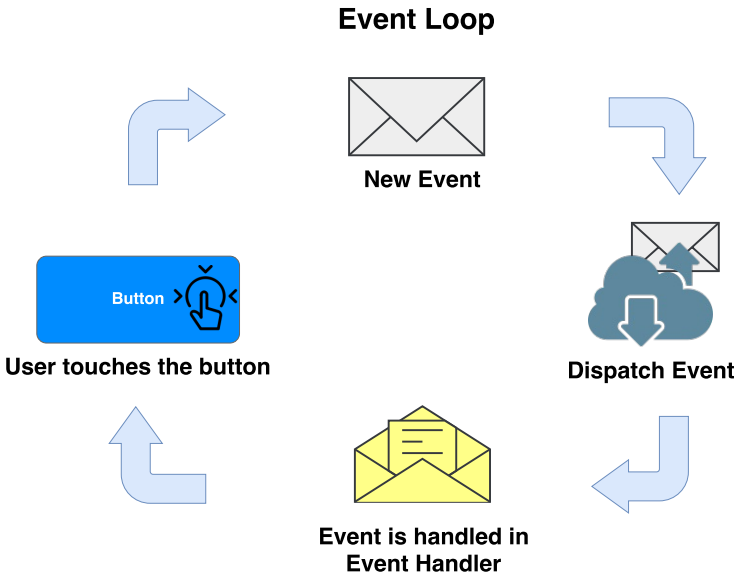
Rozhrania pre dotykové obrazovky sa v svojej podstate príliš nelíšia od ich alternatív, ktoré sú dostupné na počítačoch. Ich rozdiel spočíva hlavne v tom, akým spôsobom sú jednotlivé grafické elementy zobrazované užívateľovi. Vzhľadom na to, že jediná forma užívateľskej interakcie je vyvolaná dotykovým displejom, musia byť prvky prostredia dostatočne veľké. Ďalším rozdielom je, že rozhranie býva zobrazované na malom displeji, prípadne na obrazovke s malým rozlíšením.

Obecné požiadavky na GUI framework tvoria elementy, ktoré sú zobrazené na obrazovke. Tieto vizuálne prvky sú k dispozícii a užívateľ knižnice ich nemusí vytvárať. Element je schopný vykresliť samého seba, musí vedieť spracovať užívateľský vstup a poskytnúť spätnú väzbu.

Použitím triedneho návrhu sme schopní oddeliť spoločné vlastnosti do bázovej triedy a špecializovať konkrétne elementy. Objektový model a systém udalostí sú najdôležitejšie vlastnosti architektúry.

6.2.1 Udalosti

Široká škála frameworkov pre tvorbu GUI na desktopových systémoch dovoľuje inšpirovať sa dobrým návrhom našej knižnice. Ako bolo spomenuté, typickým a často aj jediným vstupom užívateľa je udalosť dotyku. Tieto knižnice sú bežne riadené systémom udalostí. Silno sa tu uplatňuje princíp „Event Driven Programming“. Teda, vykonávanie kódu je podmienené udalosťou ktorá sa dostane do systému.



Obr. 6.2: Cyklus spracovania udalostí v grafickej knižnici.

Vo frameworku Qt je použitý návrhový vzor Observer. Pri výskyte udalosti sú informované (notifikované) všetky entity, ktoré sa na odber danej udalosti prihlásili. Qt využíva systém signálov a slotov, ktorý nie je natívny v jazyku C++. Preto sa pred kompiláciou vykonáva predspracovanie zdrojových kódov. Signály a sloty sú nahradené callback funkciami. Princíp callback funkcií je pre Event Driven frameworky typický. Užívateľ sa nemusí zaoberať ničím iným ako implementáciou obslužnej rutiny, ktorá danú udalosť spracuje. Koncept je podobný ako pri vyvolaní prerušenia. Vtedy hardvér vyvolá obslužnú rutinu automaticky, a programátor iba napíše obsluhu udalosti.

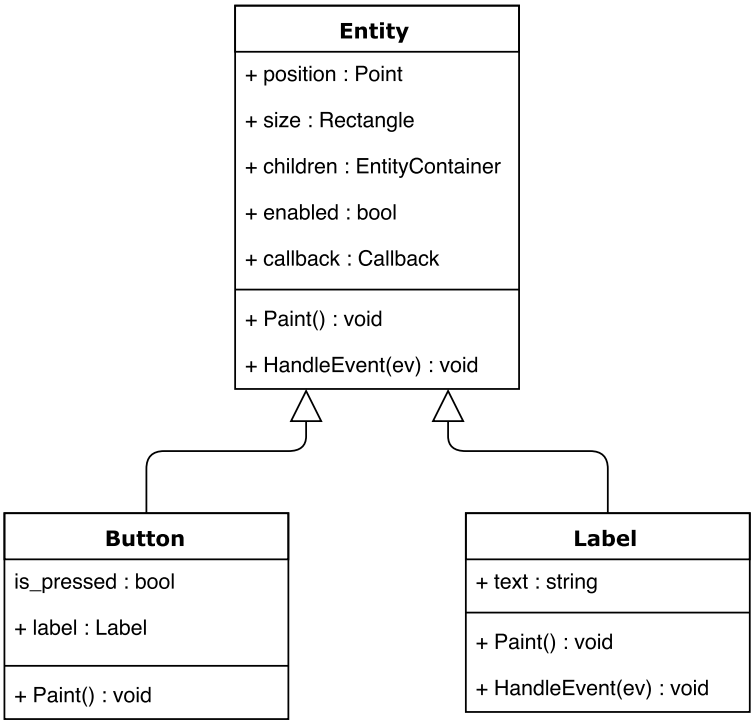
Hlavný cyklus obsluhy udalostí je zobrazený na obrázku č. 6.2. Nová udalosť sa vytvorí v momente, keď sa užívateľ dotkne obrazovky. Udalosť je prijatá do systému, ktorý dekóduje pozíciu dotyku a vyhľadá v svojich štruktúrach príslušnú obslužnú funkciu. Funkcia je zavolaná ako callback a je jej predaná udalosť na spracovanie. V tomto momente obsluha udalosti končí a celý systém čaká na ďalší užívateľský vstup.

6.2.2 Objektový model

V jazyku C++ je možné programovať objektovo a zachytiť vzťahy medzi triedami v návrhu GUI elemetov. Spoločným znakom existujúcich knižníc, ktoré sú napísané v jazyku C je model dedenia a polymorfizmu. Vývojár musí vo vlastnej réžii udržovať platnú tabuľku virtuálnych funkcií. Takýto model nezachytáva správne abstrakciu dát a funkcii, neumožňuje jednoduchú údržbu knižnice.

V našej knižnici využívame triedny systém, ktorý je bežný pre mnoho desktopových frameworkov. Bázová trieda všetkých zobraziteľných grafických elementov je trieda `Widget`.

Všetky triedy, ktoré sú od `Widget` oddedené, môžu byť vykreslené na obrazovke a prijímať správy zo systému udalostí. Výhodou tohoto modelu je, že nový grafický prvok preberá všetky funkcie, ktoré implementuje táto trieda. V praxi stačí iba predefinovať metódu vykreslovania a napísať vlastnú implementáciu spracovania udalosti.



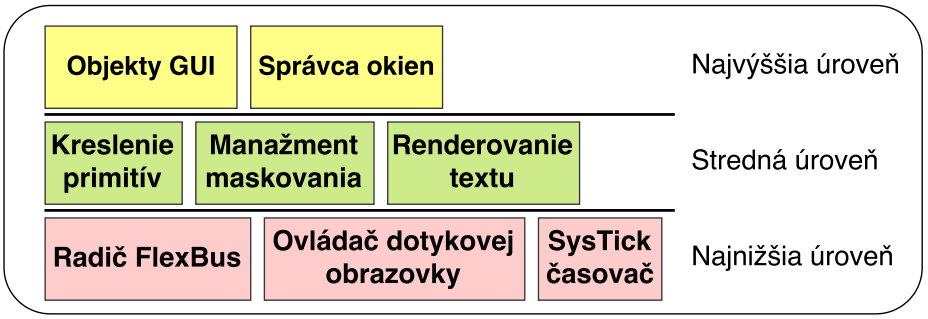
Obr. 6.3: Ilustračný triedny diagram zobrazujúci princíp vytvárania grafických prvkov v knižnici.

Princíp jednej základnej bázovej triedy je inšpirovaný frameworkom Qt. V systéme tried sa nachádza trieda `QObject`. Od tohoto objektu sú oddedené všetky ostatné grafické prvky prostredia.

6.2.3 Hierarchia úrovní

Knižnicu sme sa kvôli rozdelit do troch úrovní. Spodná úroveň je tvorená ovládačmi periférií. Stredná vrstva sa skladá prevažne z funkcií, ktoré riadia vykresľovanie pixelov, základných grafických primitív a maskovanie vykresľovanej plochy. Okrem toho sa tu nachádza knižnica pre rasterizáciu textu. Na najvyššej úrovni sa nachádza systém Widgetov, ktorý bol predstavený v predošlej podkapitole.

Na obrázku č. 6.4 je prehľadne zobrazená celá architektúra GUI systému. Každá dôležitá súčasť GUI knižnice bude popísaná v ďalších podkapitolách.



Obr. 6.4: Blokový diagram zobrazujúci 3 vrstvy knižnice.

6.3 Nízkoúrovňové ovládače

Na najnižšej úrovni sa nachádzajú ovládače periférií dotykového panelu a obrazovky. Ich úlohou je :

- Inicializácia hardvéru.
- Vytvorenie rozhrania pre vyššie úrovne knižnice.

6.3.1 Ovládač dotykového panelu

Dotykový panel na rozširujúcom module obsahuje integrovaný kontroler FT5216. Kontroler sa nachádza na ohybnom pásiku. K modulu vedie 6 kontaktov. Dva kontakty z toho sú uzemnenie a napájanie. Zvyšné vodiče sú pripojené k expanznému konektoru mikrokontroleru. Kompletný popis rozhrania kontroleru na nachádza v kapitole 3.2.1.

Kontroler je pripojený k MCU zbernicou I2C. Súčasťou zapojenia je aj resetovací signál `RST` a `INT`. Druhý menovaný je veľmi dôležitý. Pri nastavení signálu `INT` na hodnotu „0“ dáva dotykový kontroler znamenie mikrokontroleru, že došlo k udalosti na dotykovej obrazovke.

Ovládač dotykového panelu je implementovaný ako objekt triedy `TouchDriver`. V konštruktore sa nastaví pin *24* (`INT`) ako vstupný a *25* (`RST`) ako výstupný na periférii `GPIOA`. Pre `INT` sa povolí prerušenie pri klesajúcej hrane. Následne je kontroler resetovaný držaním resetovacieho signálu v nule po dobu 300 ms. Potom je inicializovaná I2C periféria na mikrokontroleri ako Master zariadenie.

Najvyššia rýchlosť komunikácie zbernice I2C, akú mikrokontroler FT5216 podporuje činí 400 000 baudov. Experimentálne sme overili, že pri vyšších rýchlostiach nie je komunikácia spoľahlivá.

Na obrázku č. 6.5 je zachytený priebeh komunikácie medzi mikrokontrolerom a kontrolerom dotykového panelu. Každá klesajúca hrana `INT` signálu indukuje, že došlo k novej udalosti dotyku. Mikrokontroler zaháji komunikáciu pomocou zbernice I2C. Samotná transakcia je veľmi rýchla oproti rýchlosti generovania nových udalostí.

Na obrázku č. 6.6 je možné vidieť detail jednej transakcie. Baudová rýchlosť činí 400 000 baud/s. Celá komunikácia trvá približne 0,2 ms.

Obr. 6.5: Zachytenie krátkeho dotyku na obrazovke. Na obrázku je zachytený priebeh signálov `INT` a I2C zbernica.

Obr. 6.6: Počas transakcie na I2C zbernici sú do mikrokontroleru načítané údaje o mieste prvého dotyku. Na obrázku sú zaznamenané dve I2C transakcie.

Pin `INT` je priradený k portu `PORTA`. Pri detekovaní zostupnej hrany dochádza k vyvolaniu prerušenia na tomto porte. Užívateľ nesmie upraviť funkciu `PORTA_IRQHandler` tak, že v nej vynechá volanie statickej metódy `TouchDriver::HandleInterrupt()`. Je dôležité poznamenať, že počas spracovávania obsluhy prerušenia nedochádza ku komunikácii s kontrolerom.

V prípade, že došlo k prerušeniu na porte `INT`, nastaví sa statická premenná indikujúca, že došlo k udalosti na dotykovom paneli. Pre testovanie tohoto príznaku slúži metóda `HasNewEvent()`. Pokiaľ je návratová hodnota nenulová potom došlo k novej udalosti. Metóda `GetEvent()` navráti udalosť v štruktúre, ktorá je určená pre spracovanie ďalšími vrstvami. V tejto metóde prebieha inicializácia I2C transakcie a parsovanie dát z odpovede. Ukážka komunikácie sa nachádza na obrázku č. 6.6.

Momentálne sa z informácií číta iba poloha prvého dotyku, napriek tomu, že kontroler podporuje až 5 simultánnych dotykových bodov. O tomto bode sa zaznamenaná poloha v osi X, Y a typ dotyku, ktorý môže byť jeden z troch:

- Udalosť dotyku prstu
- Udalosť kontaktu prstu
- Udalosť zdvihu prstu

Sekvencia udalostí je vždy v poradí: dotyk – kontakt – zdvih. Počas dlhého stlačenia obrazovky môže byť obdržaných mnoho udalostí kontaktu. Snímkovacia frekvencia dotykového panelu sa blíži až 60 Hz, to znamená že do systému môže vstúpiť až 60 dotykových udalostí za jednu sekundu.

Návratová hodnota funkcie `GetEvent()` je objekt typu `TouchEvent`. Táto trieda obsahuje všetky položky, ktoré boli uvedené v predošlom odseku.

V súčasnom návrhu sa môžu niektoré udalosti stratiť, pokiaľ mikrokontroler nestihne iniciovať dostatočne rýchlo transakciu na I2C rozhraní. Vychádza to z toho, že počas spra-

Ukážka kódu 6.2: Inicializácia I2C periférie s rýchlosťou 400 000 baudov na mikrokontroleri Kinetis

```
i2c_master_config_t masterConfig;
I2C_MasterGetDefaultConfig(&masterConfig);
// this is max baudrate, by datasheet
masterConfig.baudRate_Bps = 400000;
// I2C is clocked by bus frequency
I2C_MasterInit(I2C0, &masterConfig, CLOCK_GetBusClkFreq());
```

Ukážka kódu 6.3: Zavedenie obslužnej rutiny dotykového ovládača do programu

```
extern "C" {
void PORTA_IRQHandler(void)
{
// user defined irq handler, since we compile it as C++,
// there must be extern C
Driver::TouchDriver::HandleInterrupt();
}
}
```

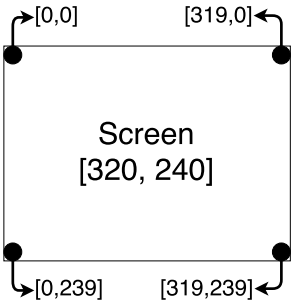
covania prerušenia sa nastavuje iba príznak výskytu novej udalosti. K samotnému čitaniu novej udalosti dochádza až o nejakú dobu. Pokiaľ sa za túto dobu zmení poloha alebo stav dotyku, stavové premenné sú v dotykovom kontroleri nahradené novými hodnotami. Riešením by mohlo byť načítanie novej udalosti a jej následné uloženie do fronty určenej k spracovaniu. Rozhodli sme sa nepoužívať frontu udalostí, pretože by to prinieslo vyššie pamätové a výkonnostné nároky na mikroprocesor počas spracovania udalosti v obslužnej rutine.

Dotykové súradnice, ktoré kontroler navráti sa vzťahujú k nulovému bodu. Ten je umiestnený v ľavom hornom rohu obrazovky. Modul netreba kalibrovať. Dotyková vrstva má pevnú pozíciu a od výroby je osadená presne. Hraničné body súradníc dotyku sú zobrazené na obrázku č. 6.7.

6.3.2 Ovládač obrazovky

Druhý nízkoúrovňový ovládač v knižnici je ovládač obrazovky, konkrétne zaisťuje prístup do video pamäti.

Ako už bolo popisované, obrazová pamäť sa fyzicky nachádza v čipe FPGA. Komunikáciu zaisťuje periféria FlexBus. Úlohou radiča je poskytnúť vyšším vrstvám knižnice komfortné rozhranie pre prístup do video pamäti.



Obr. 6.7: Pozícia súradníc dotyku na obrazovke.

Implementácia ovládača sa nachádza v triede `ScreenDriver`. K dispozícii sú metódy pre rýchle prekreslenie obrazovky jednou farbou, nastavenie farby jedného pixelu, navrátenie farby jedného pixelu.

Ovládač poskytuje dve metódy prístupu do pixelovej pamäti. Prvá varianta testuje, či zadané koordináty ležia v priestore obrazovky, a v prípade pozitívneho testu nastavia daný pixel na zvolenú farbu. Druhá varianta netestuje hranice, je implementovaná pre optimalizačné účely, kedy je z vyšších vrstiev knižnice zabezpečené, že nedôjde k vykresleniu pixelu mimo priestor obrazovky.

Inicializácia FlexBus komponenty je čiastočne prebratá z práce pána Buchty [1]. Samotná inicializácia periférie prebieha pri inicializácii pinov v súbore `pin_mux.c` je kompletne prepísaná s použitím Kinetis SDK. Parametre sú nastavené tak, aby bolo možné pristupovať do celého rozsahu DDR2 pamäti (64 MB). Pokiaľ sa v budúcnosti podarí vytvoriť rozhranie medzi FPGA a pamäťovým čipom, nebude treba upravovať tento ovládač.

Pixel je pri prenose na zbernici reprezentovaný 4 bajtovou hodnotou `Color`. Počiatočná adresa je nakonfigurovaná na 0x60000000h.

Ilustračný kód pre prácu s ovládačom obrazovej pamäte sa nachádza v ukážke č. 6.4.

Ukážka kódu 6.4: Použitie nízkoúrovňového ovládača obrazovky.

```
...
// create driver with given screen resolution
ScreenDriver driver(320, 240);
// set screen to color
driver.Clear(Colors::BLUE);
// set top left pixel [0,0] to white color
driver.PutPixel(0, 0, Colors::WHITE);
...
```

6.3.3 Meranie času

Detekcia dlhého stlačenia elementu na obrazovke je užitočná pre GUI knižnicu. Knižnica musí mať vytvorené prostredie pre meranie času. Kinetis K60 obsahuje Arm jadro. Štadardnou súčasťou tohoto jadra je podpora pre tzv. SysTick. Po vykonaní určitého počtu hodinových cyklov jadro vyvolá prerušenie a zvolá sa obslužná rutina. Typické použitie tejto vlastnosti je implementácia preemptívneho plánovača v kooperácii s operačným systémom.

V konfiguračnom súbore je nastavená taktovacia frekvencia jadra na 100 MHz. Pokiaľ nastavíme periódu opakovania na 100 000 hodinových cyklov, potom sa obslužná funkcia zavolá každú milisekundu. Toho rozlíšenie je dostatočné pre väčšinu úkonov a perióda opakovania nezaťažuje mikrokontroler.

Na globálnej úrovni je vytvorená 64-bitová premenná, ktorá slúži ako počítadlo. V každom vyvolaní obslužnej rutiny je obsah počítadla inkrementovaný. Aktuálny počet uplynutých milisekúnd od spustenia programu je možné získať pomocou volania funkcie `GetTick()`.

Je nevyhnutné aby bola premenná v jazyku C/C++ označená ako `volatile`. Pri zapnutí optimalizácií sa kompilátor snaží optimalizovať prístup do tejto premennej, ktorá je každú milisekundu inkrementovaná, bez vedomia prekladača. Pokiaľ kompilátor nevhodne optimalizuje prístup k premennej, meranie času je neúčinné. Zvolili sme 64-bitovú premennú, pretože pri použití 32-bitového počítadla a periódе 1 ms dochádza k pretečeniu za 49,71 dní. Programy na mikrokontroleroch môžu môžu byť v prevádzke roky. Preto sme sa rozhodli použiť 64-bitové rozlíšenie, ktoré už garantuje, že ku pretečeniu nedôjde. Doba, ktorá môže byť odmeraná je až 584 554 531 rokov.

Pre pozdržanie behu programu je pripravená funkcia `Delay()`. A aktívnom čakaní sa pozrdží beh programu na zadaný počet milisekúnd.

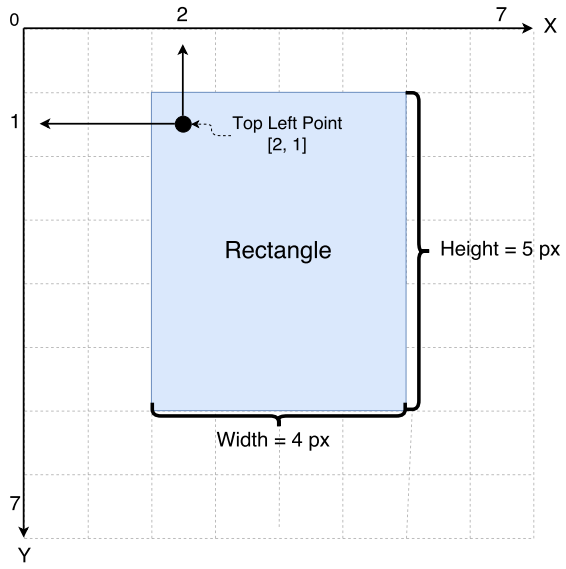
6.4 Stredná vrstva

Na strednej vrstve sa nachádzajú objekty, ktorých úlohou je odtieniť programátora od elementárnych funkcií najnižšej vrstvy. Patria sem hlavne funkcie pre kreslenie primitív na obrazovku, maskovanie oblastí, ktoré už boli vykreslené a systém pre vykresľovanie textu.

Do tejto úrovne je možné zaradiť aj pomocné podporné štruktúry `Color`, `Point` a `Rect`. Prvá menovaná je interná reprezentácia farby. Cieľom je poskytnúť jednoduché rozhranie pre reprezentáciu farby v režime RGB. Táto štruktúra poskytuje metódu, ktorou je možné z farby vytvoriť 4 bajtovú hodnotu pre prenos pomocou FlexBus rozhrania do FPGA čipu.

`Point` a `Rect` modelujú bod a štvorec v rovine. Štvorec má svoj počiatočný bod a šírku v ose X a Y. Záporná šírka alebo výška je neplatná hodnota a nesmie sa používať pre kreslenie grafických primitív. Súradnicový systém je znázornený na obrázku č. 6.8.

Z implementačného hľadiska je uvedená funkcionalita integrovaná v jedinej triede `Screen`. Táto trieda v sebe obsahuje prvok z ovládač obrazovky `ScreenDriver` a kompletne ho zapúzdruje.

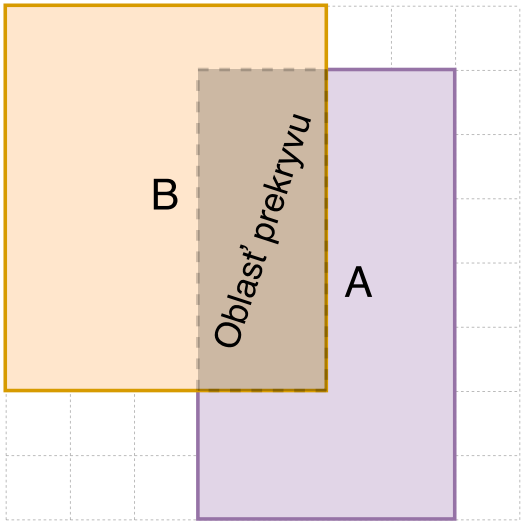


Obr. 6.8: Objekt typu `Rect` umiestnený na súradniciach [2, 6] s rozmerom [4, 5].

Jeden z problémov, ktoré knižnica na tejto úrovni rieši je proces orezávania grafických primitív pred ich vykreslením. Hlavným dôvodom, pre ktorý je potrebné riešiť orezávanie je proces prekresľovania obrazovky v prípade, že je použitý iba jeden obrazový buffer. Uvažujme, že na obrazovke sú zobrazené dva štvorce, viď obrázok č. 6.9. Objekty budú vykresľované maliarovým algoritmom. Najprv je kompletne vykreslený prvý štvorec A, potom nasleduje druhý B. Pokiaľ sa oblasti neprekrývajú, nenastáva žiaden problém. Akonáhle sa začne prelínať oblasť štvorcov, potom môžeme pozorovať obrazové artefakty v mieste prekrývania. Je to spôsobené tým, že na obrazovku je najprv umiestnený celý štvorec A. Po určitej dobe, ktorá je často väčšia ako obnovovacia frekvencia obrazovky, je do oblasti vykreslený štvorec B. Ľudské oko tento jav pozoruje ako rušivé blikanie alebo trhanie obrazu v oblasti prieniku oboch štvorcov. Tento problém nastáva výlučne pri prekresľovaní obrazovky, nie pri statickej scéne.

Jedným z riešení, ktoré je možné aplikovať, je využitie druhej obrazovej pamäti, ktorá nie je aktívna. Anglická terminológia nazýva túto metódu „Double Buffering“. Opäť sa maliarovým algoritmom rasterizujú štvorce ale až po dokončení vykresľovania sa zobrazí nový obraz na obrazovke. Kompletný obraz je zostavený mimo obrazovky. Nedochádza k vykresľovaniu nedokončeného obrazu a tým pádom ani k rušivým efektom pri prekresľovaní. Pre použitie tejto metódy musí byť alokovaný druhý obrazový buffer.

Vzhľadom na silné obmedzenie kapacity obrazovej pamäte nie je možné použiť metódu dvojitého bufferovania. V našom riešení sme sa rozhodli využiť princíp maskovania.



Obr. 6.9: Pri vykresľovaní maliarovým algoritmom dochádza k obrazovým artefaktom v prekrývajúcich sa oblastiach.

6.4.1 Orezávanie a maskovanie

Každá operácia rasterizácie grafického primitíva je najprv upravená tak, aby nedochádzalo ku vykresľovaniu mimo oblasť obrazovky. Jednoduché tvary, napríklad obdĺžnik a úsečka sú najprv transformované orezaním pomocou orezového obdĺžnika. Rozmery obrazovky sú použité pre výpočet veľkosti obdĺžnika.

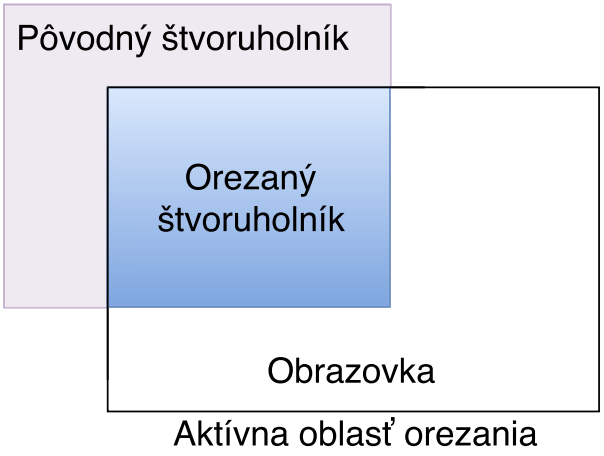
Na obrázku č. 6.10 sa nachádza obdĺžnik, ktorý je umiestnený na pozícii, kedy je na obrazovke vykreslená iba časť tohoto útvaru. Orezaním pomocou veľkosti obrazovky sme schopní odstrániť časť plochy tak, aby sa vykresľovala iba viditeľná oblasť.

Pokiaľ je grafické primitívum celou plochou mimo aktívnu oblasť, potom je celá operácia rasterizácie preskočená. V prípade, že ide o netriviálny tvar, knižnica umožňuje otestovať viditeľnosť každého pixelu, ktorý bude vykreslený na obrazovku.

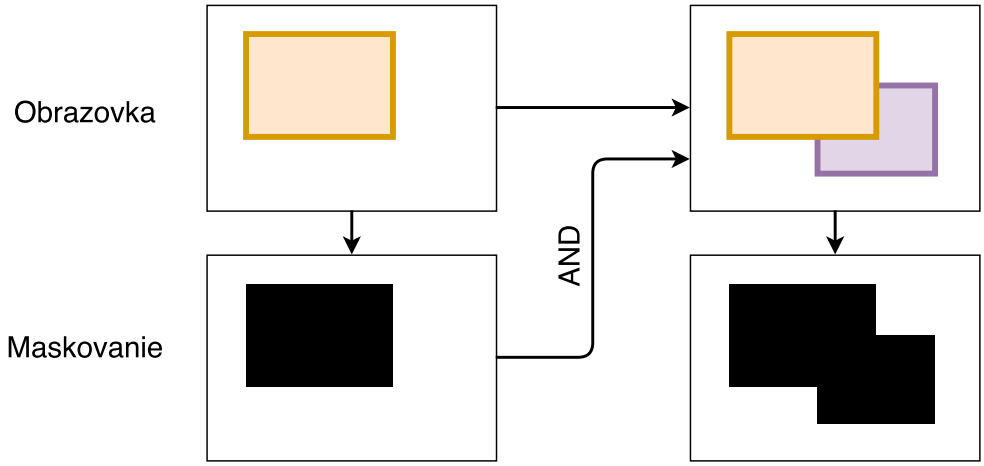
Orezávanie vykresľovania veľkosťou obrazovky nerieši problém blikania viacnásobne prekresľovaného obrazu. Je to ale prvý krok k obecnému orezávaniu ktoré je v tejto knižnici uplatnené.

V GUI prostrediach sa predpokladá, že prvky sú na obrazovke umiestnené v kontajneroch. Najvyššia entita, ktorá nie je zanorená do iného kontajneru, je okno. Na obrazovke môžu byť zobrazené viaceré okná obdĺžnikovitého tvaru, ktoré sa vzájomne prekrývajú. Každý pixel obrazovky by mal byť prekreslený najmenší možný počet krát. Ideálne je dosiahnuť stav, pri ktorom sa pixel prekreslí iba raz.

Pre riešenie viditeľnosti okien sa využíva metóda maskovania. Vykresľovanie okien prebieha od najvrchnejšieho po najhlbšie, teda opačným smerom ako vykresľuje maliarov algoritmus. Na začiatku algoritmu je maska nulová. Po vykreslení prvého okna sa v maske nastavia všetky pixely patriace oknu na hodnotu „1“. Tento stav je zobrazený na obrázku č. 6.11. Pri vykresľovaní ďalšieho okna sa testuje, či daný pixel už bol aktualizovaný. Pokiaľ



Obr. 6.10: Orezávanie grafických primitív dimenziou obrazovky.

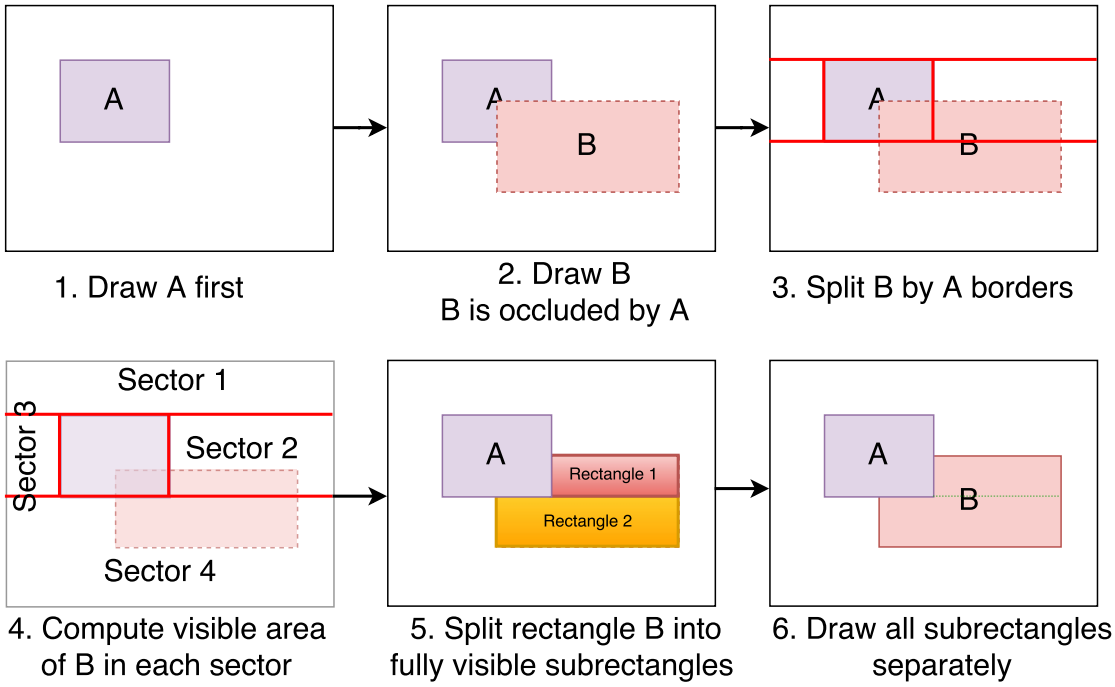


Obr. 6.11: Princíp maskovania grafických operácií.

sa v pamäti masky pre daný pixel nachádza hodnota „1“, potom daný pixel už bol vykreslený a nesmie byť znovu modifikovaný. Tento proces prebieha pre všetky okná, ktoré majú byť zobrazené.

Podobný princíp sa uplatňuje na grafickej karte. Avšak operácie sú hardvérovo akcelerované. Tento spôsob by bolo môžné implementovať na FPGA v prípade, že by to kapacita video pamäte umožnila. V prípade výpočtu na mikrokontroleri je neefektívne vykonávať pri každom vykresľovaní testovanie viditeľnosti na pixelovej úrovni.

Lepšie riešenie je rozdeliť okno na viacero častí tak, že výsledné časti nebudú prekryté iným nadradeným oknom. Bitovú masku nahradíme kontajnerom, ktorý bude obsahovať popis plochy, ktorá už bola prekreslená. Pre popis pravouhlej oblasti použijeme triedu `Rect`. Po každom vykreslení okna sa do interného kontajneru uloží nová položka reprezentujúca práve vykreslené okno. Pri vykresľovaní nasledovného okna sa rešpektujú všetky oblasti, ktoré sa v kontajneri nachádzajú. Vykresľovanie je umožnené iba do prázdnej plochy.



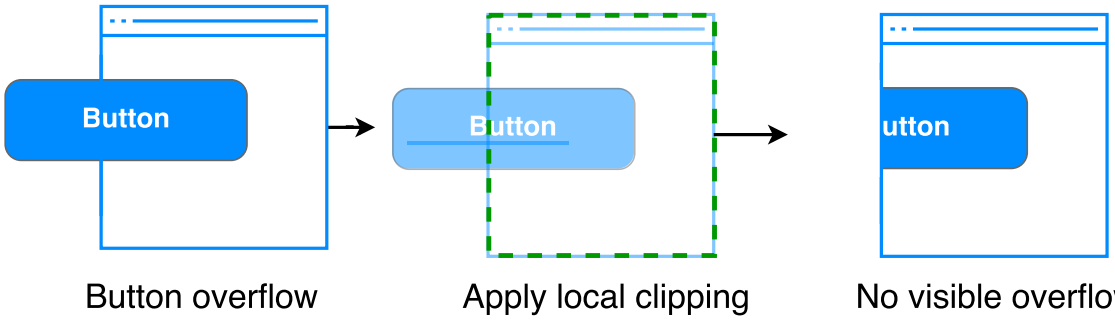
Obr. 6.12: Princíp riešenia viditeľnosti na základe maskovania, ktorý sa používa v tejto knižnici.

Detailný popis vykresľovania dvoch obecných prekrytých obdĺžnikov sa nachádza na obrázku č. 6.12. Útvar A je vykreslený ako prvý, pretože ja nachádza na najvyššej úrovni. Po vykresneí objektu A sa uloží do interných štruktúr knižnice informácia o pozícii a veľkosti okna. Pri vykresľovaní obdĺžniku B sa zistí, že je čiastočne prekrytý objektom A. V kroku 3 a 4 sa rovina rozdelí na 4 oblasti pomenované `Sector`. V týchto oblastiach sa pomocou algoritmu orezávania rozdelí objekt B na menšie časti. V kroku 5 sú zvýraznené 2 obdĺžniky, ktoré vznikli z pôvodného objektu B. Je garantované, že výsledné útvary sú viditeľné a preto môžu byť vykreslené bez dodatočného testovania viditeľnosti.

Algoritmom pre vykresľovanie zakrytých oblastí je možné efektívne riešiť viditeľnosť objektov v prípade, že nie je možné použiť maliarov algoritmus. Z pamäťového hľadiska sa musia pre každý vykreslený objekt udržovať metadáta o polohe a veľkosti. Pri výpočte neprekrývajúcich sa oblastí môže vzniknúť niekoľko výsledných pravouhlých objektov v závislosti na prekrytej oblasti.

Algoritmus pracuje výlučne s pravouhlými oblasťami. Rozdeľovanie obecných a polygonálnych tvarov nie je implementované. Obecné tvary by vyžadovali mnoho operácií s plávajúcou desatinnou čiarkou, čo je v prípade mikrokontroleru príliš veľké spomalenie.

V implementácii triedy `Screen` sa nachádza štandardný C++ šablónový kontajner `std::vector<Rect> clippingStack`. Tento vektor v sebe udržuje všetky oblasti, ktoré sú maskované. Vo verejnom rozhraní sa nachádza metóda na resetovanie orezávacích nastavení `ResetClipping()`. Nová maska je do systému zavedená pomocou volania `ClippingStackAdd()`.



Obr. 6.13: Orezávacia oblasť môže byť nastavená na rozmer okna aby sa vnútorné elementy nedostali mimo plochu okna.

V prípade grafických operácií je často potrebné definovať oblasť, do ktorej budú vykresľované grafické primitíva. Modelová situácia je ilustrovaná na obrázku č. 6.13 . Vykresľuje sa okno s tlačidlom, ktoré sa nachádza na takej pozícii, že polovica tlačidla zachádza za okraj okna. Bez nastavenia orezovej oblasti by tlačidlo zaberalo aj plochu mimo okna tak, ako je zobrazené na prvej časti obrázku. Pomocou funkcie `LocalClippingMask()` je možné obmedziť aktívnu oblasť kreslenia na definovaný obdĺžnik. Samozrejmosťou je rešpektovanie maskovaných oblastí.

Z hľadiska implementácie je dôležité, aby operácie orezávania a viditeľnosti boli čo najrýchlejšie. Časová zložitosť algoritmu testovania viditeľnosti pixelu narastá lineárne s počtom platných viditeľných oblastí vo vektorovom kontajneri, ktorý udržuje tieto informácie. Prienik bodu s pravouhlou oblasťou má zložitosť maximálne 4 aritmetických porovnaní súradníc. Pri výpočtoch sa používa pevná desatinná čiarka.

6.4.2 Grafické primitíva

Na strednej vrstve knižnice sa nachádzajú všetky funkcie, ktoré rasterizujú grafické primitíva. Základné objekty podporované v knižnici sú bod, čiara, obdĺžnik a kruh. Objekt je možné vykresliť na obrazovku na ľubovoľnej pozícii X a Y. Každý objekt je vykreslený zvolenou farbou. Pre obdĺžnik a kruh exuistujú varianty funkcií, ktoré vykreslia iba obvod telesa.

V počítačovej grafike sa pre rasterizáciu grafických primitív používajú najčastejšie efektívne algoritmy, ktoré boli optimalizované pre výpočet nad celými číslami. Matematický základ pre algoritmy, ktoré sú použité v tejto knižnici, je čerpaný hlavne z publikácie [4].

Vykreslenie bodu je najjednoduchší typ operácie. V tomto prípade stačí iba indexovať video pamäť na zadaných súradniciach a priamo nastaviť farbu pixelu.

Vykresľovanie úsečky je implementované Bressenhamovým algoritmom. V referenčnej publikácii sa popis začína v kapitole 3 na strane 88. Čiara medzi dvoma bodmi je široká práve 1 pixel a môže byť ľubovoľnej farby. V knižnici nie je podporované kreslenie čiary s inou veľkosťou pera. V prípade potreby môže byť vykreslenie takej čiary byť implementované tak, že sa pomocou Bressenhamovho algoritmu určia body na úsečke a v každom

bode sa vykreslí kruh s priemerom šírky čiary a danou farbou. Toto riešenie však nie je optimálne, pretože by dochádzalo k viacnásobnému prekresľovaniu oblastí čiary, ktoré už boli zobrazené. V tomto prípade nebude dochádzať k blikaniu alebo trhaniu obrazu ale k zbytočnému počítaniu výplne kruhu.

Rasterizácia obvodu obdĺžnika je implementovaná vykreslením 4 obvodových čiar okolo plochy útvaru. Takáto implementácia dovoľuje v budúcnosti jednoducho pridať podporu pre kreslenie ľubovoľného štvoruholníka.

Rasterizácia obdĺžnika, ktorý je vyplnený farbou, je vytvorená pomocou dvoch vnorených `for` cyklov. Idea je založená na princípe *ScanLine* algoritmu. Pred vykreslením dochádza dochádza k odstráneniu oblastí, ktoré nebudú vykreslené pomocou algoritmu popísaného na obrázku č. 6.12. Vykresľovanie prebieha od ľavej hrany obdĺžnika po pravú hranu. Postupnosť riadkov je od najvyššieho po najnižší.

Najzložitejšou funkciou je rasterizáciu kruhu a kružnice. Obe metódy sú implementované podľa Bressenhamovho Midpoint algoritmu (str. 97, [4]). Kruh a kružnica je definová stredovým bodom, polomerom a farbou výplne resp. farbou čiary. Testovanie viditeľnosti a orezávanie musí byť vykonané pre každý pixel zvlášť, pretože pri použití týchto algoritmov sa nedá iným spôsobom obmedziť vyplňovaná oblasť v závislosti na maskovaní.

Pokročilé grafické funkcie ako je priehľadnosť a farebné gradienty v tejto knižnici nie sú implementované vzhľadom na malý farebný priestor.

6.4.3 Rasterizácia textu

O zobrazovanie fontov sa stará knižnica MCUFont [11]. Projekt je udržiavaný na GitHube. Rozhodli sme sa využiť knižnicu, ktorá sa používa aj v komerčnom riešení μGFX. Táto knižnica je napísaná v jazyku C. Podporuje formáty TTF a BDF. Podľa popisu autorov má po kompilácii približne 3 kB.

Knižnica je rozdelená na dve časti. Prvá časť je určená pre PC a stará sa o prevod súboru s fontom do internej repezentácie. Druhá časť knižnice beží na mikrokontroleri, číta enkódovaný font a vykresľuje zvolený text.

Prevod fontov prebieha v enkódovacom programe na počítači s operačným systémom Linux. Zo vstupného súboru je vygenerovaný kód v jazyku C, ktorý obsahuje komprimované dáta fontu. Tento súbor musí byť pridaný do kompilácie počas prekladu kódu pre mikrokontroler.

Knižnicu sme integrovali do triedy `Screen` a úplne sme ju zapúzdrili. Vzhľadom na rozhranie knižnice sme museli do triedy pridať podporné funkcie pre udržiavanie kontextu vykresľovania.

Metódy, ktoré sú určené pre vykresľovanie textu na obrazovke sú doplnené o možnosť zarovnania textu v horizontálnom aj vertikálnom smere. Okrem toho je možné text vykresliť do priestoru definovaného pomocou objektu `Rect`. V tomto prípade je knižnica automaticky schopná cieľový text rozdeliť na viac riadkov, pokiaľ by sa do zvolenej oblasti nezmestil celý v jednom riadku.

Implementácia adaptéru pre rozhranie jazyka C sa nachádza v triede `Screen`. Dostupné užívateľské funkcie pre vykreslenie jedného riadku textu sú nasledovné:

- `DrawText()` – Funkcia preberá text, pozíciu X a Y farbu textu a zarovnanie v horizontálnom smere.
- `DrawTextRect` – Funkcia preberá text, oblasť `Rect` a zarovnanie v horizontálom a vertikálnom smere.

Počas vývoja sme využívali malé rastrové písmo `fixed_5x8`. Definícia sa nachádza vo vygenerovanom súbore, ktorý je dodaný k zdrojovým kódom knižnice.

Všetky písma, ktoré sa v knižnici momentálne nachádzajú sú uvedené v tomto zozname:

- DejaVuSans12 – písmo s podporou priehľadnosti a veľkosťou 12
- DejaVuSans12bw – varianta DejaVuSans12 bez podpory priehľadnosti
- DejaVuSerif16 – DejaVuSans s veľkosťou fontu 16
- DejaVuSerif32 – DejaVuSans s veľkosťou fontu 32
- fixed_10x20 – rastrové písmo s pevnou veľkosťou znaku
- fixed_5x8 – rastrové písmo s pevnou veľkosťou znaku
- fixed_7x14 – rastrové písmo s pevnou veľkosťou znaku

Súradnice X a Y, prípadne definícia oblasti `Rect`, v ktorej má byť text vykreslený, sú definované v globálnom priestore obrazovky. Samozrejmosťou je podpora pre maskovanie a orezávanie. Zaujímavou vlastnosťou knižnice je podpora pre anti-aliasing. V našom prípade túto funkciu nie je možné použiť, pretože nie je možné správne zmiešať farby pixelov.

6.5 Najvyššia úroveň

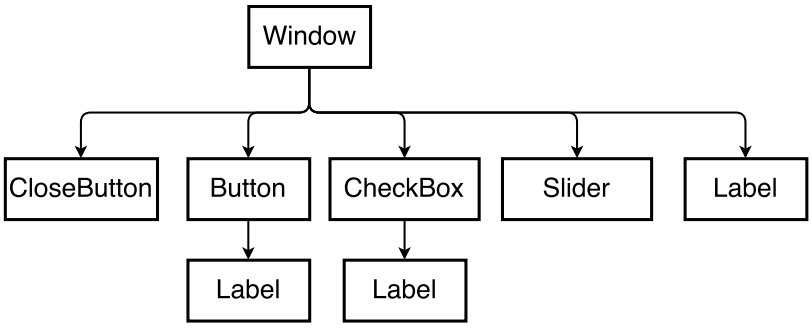
Na najvyššej úrovni tejto knižnice sa nachádza systém Widgetov a systém predávania udalostí `Event`.

6.5.1 Widgety

`Widget` je spoločná bázová trieda pre objekt, ktorý môže byť vykreslený na obrazovku. Takýto objekt môže byť tlačidlo, textové pole, posuvník, okno. Väčšina spoločných vecí je implementovaná v bázovej triede. Oddedené triedy musia implementovať metódu `Paint`, pretože v bázovej triede je definovaná ako „pure virtual“ metóda. Táto funkcia je určená na vykreslenie daného Widgetu. Okrem toho môžu reimplementovať štandardné funkcie pre reagovanie na vstupné udalosti.

Základný atribút Widgetu je poloha, respektíve obalové teleso – bounding box. Tento priestor je definovaný ako plocha `Rect`. a súčasne definuje rozmery Widgetu. Používa sa pre systém vykresľovania a pre hľadanie objektu, ktorému bude doručná udalosť `TouchEvent`.

Každý Widget môže mať svojho rodiča a niekoľko potomkov. Vzniká tak stromová reprezentácia logického priestoru na obrazovke tak ako je možné vidieť na obrázku č. 6.14.



Obr. 6.14: Stromová architektúra systému widgetov pre jednoduché okno.

Widget obsahuje príznaky, ktoré definujú jeho schovanie alebo určujú jeho stav. Sú implementované ako bitové pole `std::bitfield`. Popis týchto sa nachádza v tabuľke č. 6.1.

Tabuľka 6.1: Príznakové bity v triede `Widget`.

Názov položky	Sémantika
`ENABLED`	Definuje stav, či je widget aktívny alebo neaktívny
`PASS_EV_TO_PARENT`	Pasivácia widgetu. Všetky udalosti, ktoré sú doručené budú zaslané rodičovi
`VISIBLE`	Definuje či je widget viditeľný
`PRESSED`	Príznak je nastavený iba v prípade aktívneho dotyku
`TRANSPARENT`	Indikuje, že widget na svojej ploche obsahuje priehľadnú plochu ktorá musí byť vykreslená z nadradedného Widgetu
`DIRTY`	Tento príznak zabezpeči, že sa widget nanovo vykreslí v dalšom zaslané vykresľovania
`GEOMETRY_CHANGED`	Udáva stav, kedy došlo k smene obaľovacieho telesa bounding box. Takýto widget musí byť nanovo vykreslneý spolu s rodičovským widgetom
`DELETE`	Widget bude odstránený zo stromu widgetov.

Vykresľovanie widgetov prebieha iba v prípade, že došlo k zmene vizuálneho stavu a objekt musí byť vykreslený nanovo. V tomto prípade sa testuje kompletná stromová štruktúra. V prípade, že Widget označený príznakom `DIRTY`, je vykreslený odznova celý. Avšak, v prípade, že takýto Widget bol označený príznakom `TRANSPARENT`, potom musí byť vykreslený nanovo aj rodičovský Widget. Je to z toho dôvodu, že takýto widget je vykreslený na pozadí rodičovského elementu a iba pri prekreslení vnoreného objektu nemusí byť garantované, že sa aktualizujú všetky časti výsledného obrazu.

Jedným z dôležitých príznakov je `GEOMETRY_CHANGED`. Ak je nastavený, potom došlo k zmene obaľovacieho telesa Widgetu. Poloha alebo veľkosť Widgetu bola zmenená. V tomto prípade je nutné podobne ako pri `TRANSPARENT` nanovo vykresliť celý rodičovský objekt.

V prípade, že programátor chce niektorý Widget nastaviť ako neviditeľný, musí tak spravit pomocou príznaku `VISIBLE`. Widget v tomto móde je ignorovaný počas prekresľovania obrazu aj počas delegovania udalostí. Aby bolo možné Widget zobrazit ale zablokovať iba príjem udalostí je potrebné nastaviť príznak `ENABLED` na `false`. V prípade, že je vhodné aby namiesto objektu preberal preberal udalosti jeho rodičovský prvok, je potrebné zapnúť príznak `PASS_EV_TO_PARENT`. Tento stav je typický napríklad pre tlačidlo `Button`, kedy je v strede tlačidla zobrazený text `Label`. Po stlačení textového objektu je potrebné delegovať udalosť na nadradený element, inak by nedošlo k aktivácii udalosti stlačenia.

V knižnici sa nachádza základný výber Widgetov:

- `Label` – textové pole
- `Button` – štandardné tlačidlo
- `CheckBox` – zatrhávacie tlačidlo
- `RadioButton` – Radio tlačidlo, vždy je aktívny iba jeden objekt zo skupiny
- `HorizontalSlider` – vodorovný posuvník
- `Window` – Widget, ktorý reprezentuje okno na obrazovke

Názvy tried Widgetov sú zhodné s označením, ktoré sa používa vo frameworku Qt. Ich význam a sémantika je rovnaká. Od týchto tried je možné dediť a vytvoriť tak vlastné komponenty. Napríklad zvoliť si vlastný typ vykresľovania tlačidla alebo zmeniť jeho chovanie na určité typy udalostí dotyku.

Je potrebné poznamenať, že objekt triedy `RadioButton` musí byť pridaný do špecializovaného kontajneru, ktorý rieši aktiváciu a deaktivíciu ostatných objektov v tejto skupine. Názov tohoto kontajneru je `Radiogroup`.

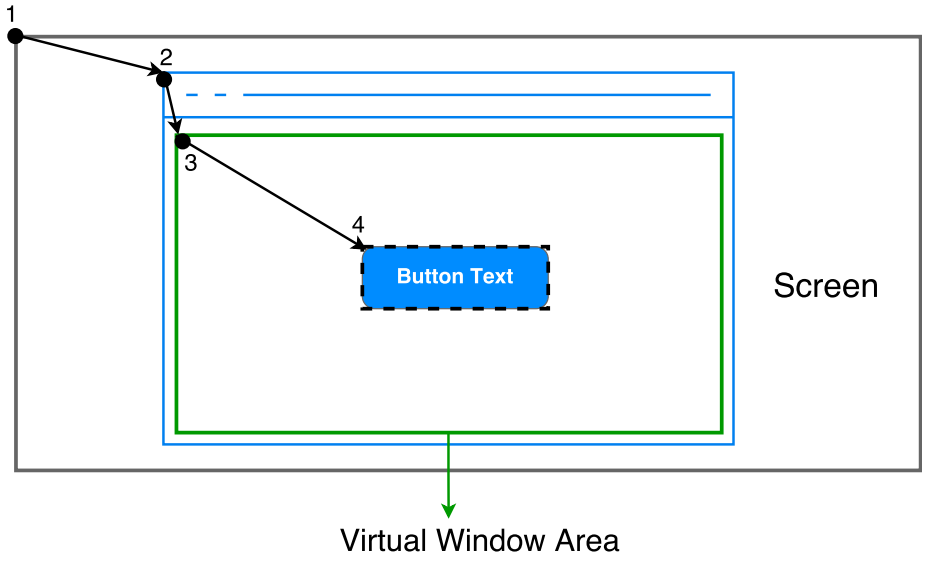
Z hľadiska vykresľovania je zavedený systém kedy nadradený widget definuje pomocou offsetu oblasť, do ktorej je možné vykresľovať dcérske objekty. Koordináty vo vnútri oblasti sú lokálne vzhľadom na jej začiatok. Praktický význam spočíva v jednoduchom prekresľovaní widgetu pri zmene pozície nadradeného elementu. Napríklad pri posuve okna sa automaticky prekreslia objekty okna na pozíciách relatívnych k zvolenému oknu. Druhým praktickým dôvodom je jednoduché vyhľadanie objektu pre umožnenie delegovania udalostí.

Z pohľadu implementácie je potrebné uvedomiť si, že stredná vrstva knižnice pracuje s globálnym súradnicovým systémom a neprekladá lokálne dimenzie objektov zanorených v strome Widgetov. Pri kreslení musí objekt sám zabezpečiť, aby kresliacim funkciám predal globálne súradnice. To je umožnené sadou translačných funkcií, ktoré implementuje trieda

`Widget`. Posuv súradníc prebieha pripočítavaním lokálnych súradníc logických oblasti – offsetov.

6.5.2 Systém udalostí

Spôsob delegovania udalostí je vo svojej podstate inverzný k metóde vykresľovania widgetov na obrazovku. Proces vyhľadania objektu, ktorému bude zaslaná udalosť funguje nasledovným spôsobom. V nasledujúcej ukážke predpokladáme, že na obrazovke sa nachádza okno s jediným tlačidlom v usporiadaní, ktoré je znázornené na obrázku č. 6.15. Ďalej predpokladajme, že užívateľ sa práve dotkol tlačidla. Na obrázku sú číslami vyznačené dôležité body, ktoré sú používané v krokoch algoritmu. Bod s číslom 1 označuje počiatok súradnicového systému.



Obr. 6.15: Proces prekladu globálnej súradnice na lokálnu súradnicu.

Proces delegovania začína na najvyššej úrovni. Na tejto úrovni spravuje okná manažér okien. Pri vstupe udalosti do systému musí správca vybrať správne okno podľa globálnych súradníc dotyku okno. Tomuto oknu sa budú doručovať všetky nasledovné udalosti až pokým do systému nevstúpi správa o zodvihnutí prstu.

V momente, kedy udalosť vstupuje do okna, je na základe bodu 2 upravená globálna pozícia dotyku. Od globálnej súradnice sú odčítané zložky bodu 2. V okne sa testuje, či udalosť patrí oknu alebo do oblasti, kde sa vykresľujú dcérske objekty. Oblasť je na obrázku označená názvom „Virtual Window Area“.

V modelovej situácii dochádza k delegovaniu udalosti do vnútra virtuálnej oblasti. Preto je súradnica opäť upravená odčítaním bodu 3. Dôležité je poznamenať, že bod 3 v sebe obsahuje iba posuv od bodu 2 a nie globálnu súradnicu, ako by sa mohlo z obrázku zdať.

Vo virtuálnej oblasti je podľa pozícii obaľovacích telies widgetov vyhľadané tlačidlo. Následne je opäť upravená súradnica podľa bodu 4. Vo funkcii, ktorá spracováva udalosti

tlačidla je dotyková súradnica reprezentovaná ako posuv od bodu 4. Celý systém delegovania udalostí je založený na rovnakom princípe ako mapovanie lokálnych súradníc na koordináty obrazovky.

V triede `Widget` je implementované rozhranie pre správu prichádzajúcich udalostí. Výňatok z hlavičkového súboru je zobrazený v ukážke č. 6.5. Aby užívateľ nemusel pre každý Widget písať vlastné chovanie, stačí napísať implementáciu pre prvé 4 funkcie v ukáže. Implementácia poslednej metódy `HandleTouchEvent` na základe typu udalosti a merania uplynutého času jednu z prvých 4 funkcií. Význam funkcií je popísaný v tabuľke č. 6.2. Dlhé stlačenie je detekované pomocou systému merania času. V štandardnom nastavení je časová doba pre určenie dlhého dotyku nastavená na 400 ms.

Ukážka kódu 6.5: Rozhranie pre manažment udalostí

```
virtual void HandleLongDown(const TouchEvent& ev);
virtual void HandleDown(const TouchEvent& ev);
virtual void HandleUp(const TouchEvent& ev);
virtual void HandleContact(const TouchEvent& ev);
virtual void HandleTouchEvent(const TouchEvent& ev);
```

Tabuľka 6.2: Význam funkcií z rozhrania pre správu udalostí.

Názov metódy	Popis
`HandleLongDown`	Vyvolané pri dlhom stlačení widgetu
`HandleDown`	Vyvolané pri stlačení widgetu
`HandleUp`	Vyvolané pri pustení widgetu
`HandleContact`	Vyvolané v prípade, že došlo k stlačeniu a posuvu prstu
`HandleTouchEvent`	Vyvolané pri každom doručení udalosti

Každý widget je možné asociovať s funkciou, ktorá bude zavolaná v prípade, že dôjde k aktivácii widgetu. Napríklad, pre tlačidlo sa funkcia zavolá v prípade, že dôjde k jeho stlačeniu, dlhému stlačeniu alebo pusteniu. Príklad nastavenia callback funkcie je zobrazený na ukážke č. 6.7.

6.5.3 Správca okien

V hierarchii Widgetov je na najvyššej úrovni umiestnené okno `Widget`. Okná spravuje manažér `WindowManager`. Jeho úlohou je zobrazovať okná na LCD obrazovku. Okrem toho rieši vykresľovanie okien a správu oblastí tým že definuje masky a orezové plochy pre nižšie úrovne knižnice.

Ukážka kódu 6.6: Príklad nastavenia Callback funkcie pre jedno tlačidlo

```
class ExampleClass
{
ExampleClass()
{
...
auto btn = new Button(this, {{0,0}, {80, 30}}, "Button");
btn->SetCallback(CALLBACK(ExampleClass::CallbackFunction));
...
}
void CallbackFunction(const CallbackEvent& ev)
{
if (ev.GetState() == CallbackEvent::Down)
{
// handle button press
}
}
```

Správca okien tvorí hlavný vstup dotykových udalostí do systému. Správca vyhľadá okno, ktoré leží pod bodom dotyku a preposiela všetky udalosti oknu až dokým nedôjde k udalosti ukončeniu dotyku.

Pri posuve okien po obrazovke sa detekujú neplatné oblasti a okná, ktoré musia byť vykreslené, aby došlo k správnemu prekresleniu obrazovky. Všetky okná, ktoré majú byť na obrazovke zobrazené musia byť v správcovi registrované.

Po každom spracovaní jednej udalosti sa v tomto objekte kontroluje stav Widgetov. Pokiaľ je pomocou príznaku `DELETE` detekované, že niektorý widget má byť zmazaný, spustí sa procedúra odstraňovania Widgetov.

6.5.4 Event Loop

Hlavný cyklus programu je v prípade systémov udalostí veľmi podobný. Čaká sa v cykle, kým udalosť vstúpi do systému. Potom je objekt udalosti predaný do správu okien, kde sa vykoná delegácia udalosti a zavolá sa príslušná obslužná funkcia. Po návrate z obsluhy sa zavolá správca okien znovu s požiadavkom na aktualizáciu obrazu na obrazovke. Pokiaľ k žiadnej zmene nedošlo, potom je táto operáciu bez efektu. Inak sa aktualizujú neplatné časti obrazu na obrazovke.

Ukážka kódu 6.7: Príklad nastavenia Callback funkcie pre jedno tlačidlo

```
// Main event loop
while (true)
{
// this just waits for a new touch event, but user may
// implement custom event loop
if (t.HasNewEvent())
{
// once we got new touch points
const TouchEvent e2 = t.GetEvent();
// let it pass to manager to find corrent widget
manager.HandleEvent(e2);
// and redraw screen if something has changed
manager.Update();
}
}
```

V ukážke č. 6.7 je uvedený hlavný cyklus, ktorý sa používa v demonštračnej aplikácii našej knižnice. V prípade iných projektov je pravdepodobné, že hlavný cyklus Event Loop bude zhodný.

Kapitola 7

Zhodnotenie a vylepšenia

V súčasnom stave rozširujúci modul limituje GUI knižnicu tým, že k dispozícii je iba 64 farieb. Napriek veľkému úsiliu sa neporadilo oživiť komunikáciu s DDR2 čipom. Na základe tejto skutočnosti je do budúcnosti potrebné integrovať dosku na rozširujúceho modul pamäťový čip. Počet voľných kontaktov je 11. To v súčasnom zapojení neumožňuje pripojiť SRAM pamäť.

Naskytajú sa dve riešenia. Prvým riešením je integrovať na dosku kontroler TFT obrazovky. V tom prípade je lepšia alternatíva nahradiť celý TFT modul iný modelom. Podľa stránok najväčších distribútorov súčiastok sa totiž kontroler samostatne kúpiť nedá.

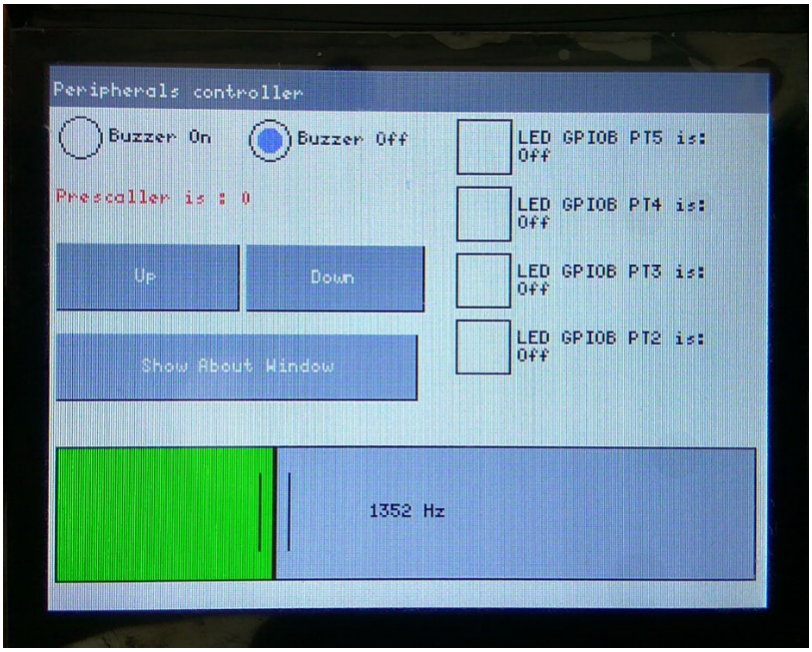
Táto varianta by pravdepodobne obmedzila rýchlosť prenosu dát na displej. Niektoré kontrolery môžu byť riadená RGB signálom ale z dokumentácie nie je jasné, či sa v tomto režime používa integrovaná video pamäť pre bufferovanie obrazu.

Druhým riešením je postaviť rozširujúci modul s vlastným prídavným FPGA čipom. Táto varianta je univerzálnešia a flexibilnejšia. K FPGA čipu by bola pripojená vlastná SRAM pamäť. V tejto konfigurácii by bolo možné implementovať vlastný grafický akcelerátor s množstvom pokročilých funkcií. Veľká nevýhoda by spočívala vo väčšej cene za jeden kus rozširujúceho modulu.

Ako bolo napísané, knižnica trpí tým, že nie je možné zobraziť kvalitnú rasterovú grafiku. Návrh knižnice bol tomu podriadený. V implementácii chýba podpora pre rasterové obrázky alebo farebné gradienty.

Systém Widgetov založený na obaľovacom telese s obdĺžnikovým tvarom. Vylepšením systému môže byť pridanie telesa s kruhovým tvarom, prípadne tvarom, ktorý je definovaný polygónom. Rovnako tak môže byť vytvorená fronta vstupných udalostí, pretože pokiaľ bude trvať spracovanie požiadavky v systéme dlhšiu dobu, potom sa strácajú udalosti dotyku.

Jazyk C++ sa ukázal byť dostatočne výkonný pre použitie na mikrokontroleri. Demoaplikácia beží pomerne rýchlo. Podľa profilovania kódu trvá najdlhšie výpočet orezávania a maskovania pri pixelových operáciách. Pri kompilovaní na najvyššom stupni optimalizácie O3 je možné všimnúť si veľké zrýchlenie behu aplikácie.



Obr. 7.1: Demo-aplikácia napísaná nad GUI knižnicou. Obrazovka umožňuje ovládať LED diódy a riadiť bzučiak. Pruhy na obrazovke sú spôsobené aliasingom na strane fotoaparátu.

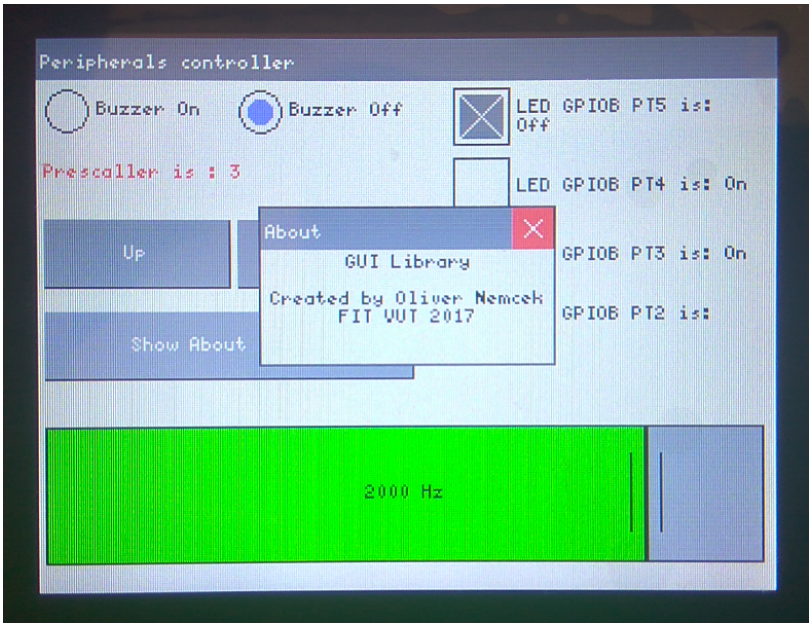
Pre demonštračné účely bola vytvorená demo aplikácia, ktorá umožňuje riadiť periférie na mikrokontroleri. Program je napísaný nad GUI knižnicou a demonštruje všetky Widgety, ktorými knižnica v momentálne disponuje. Zdrojový kód je komentovaný a tak je možné na základe tohoto kódu pochopiť základné princípy pre používanie knižnice. Po spustení sa zobrazí na displeji okno, sada tlačidiel, horizontálny posuvník, textové pole. Okrem toho je demonštrované ako zobraziť modálne okno pomocou tlačidla *Show About Window*.

Kód demo-aplikácie zaberá približne 45 kB pri kompilácii s parametrom O3. Jedným z potrebných krokov, pre úspešné spustenie aplikácie je zvýšenie pamäťového rozsahu pre haldu na približne 15 kB.

Tabuľka 7.1: Prehľad použitia pamäťových regiónov po kompilácii demo-aplikácie.

Pamäťový región	Použítá veľkosť	Celková veľkosť	Použitá pamäť [%]
PROGRAM_FLASH	47256 B	512 kB	9,01 %
SRAM_UPPER	16912 B	64 kB	25,81 %
SRAM_LOWER	0 B	64 kB	0,00 %

Metriky zdrojových kódov knižnice sú uvedené v nasledovnej tabuľke. Knižnica MCU-Font je pripočítaná k číslam, pretože je dodávaná spolu so zdrojovými kódmi knižnice



Obr. 7.2: Po stlačení tlačidla *Show About Window* sa objaví druhé okno, ktorým je možné pohybovať.

Tabuľka 7.2: Metrika zdrojových kódov knižnice GUI

Typ súboru	Počet súborov	Počet riadkov kódu
C	19	4605
C++	15	2062
C/C++ Header	29	994
Spolu	63	7661

Kapitola 8

Záver

V tejto práci sme sa venovali návrhu a konštrukcii rozširujúceho LCD modulu, ktorý bude pripojený k výukovej doske Minerva. Vybrali sme TFT LCD displej s kapacitnou dotykovou vrstvou. Veľkosť uhlopriečky displeja je 3,5" a rozlíšenie je 320x240 obrazových bodov. Cena jedného LCD modulu sa pohybuje na úrovni približne $40.

Prototyp plošného spoja bol navrhnutý v programe Altium Designer. Doska bola vyrobená a osadená súčiastkami. Drobné komplikácie s hardvérom boli úspešne vyriešené. Modul je po hardvérovej stránke plne funkčný.

LCD displej musí byť pripojený na digitálne 24-bitové RGB rozhranie. Riadiaca signalizácia na rozhraní vychádza z analógového VGA signálu. Obraz na displeji musí byť obnovovaný približne 60-krát za sekundu inak dochádza k strate farieb. Kontroler RGB rozhrania bol umiestnený do logiky FPGA čipu. Implementácia zaberá minimálne množstvo prostriedkov na FPGA.

Počas riešenia práce sa objavili neočakávané komplikácie s komunikáciou medzi FPGA a DDR2 čipom. Na tomto pamäťovom čipe mala byť uložená video pamäť. Miesto pre uloženie muselo byť vyhradené priamo na FPGA čipe. Z tohoto dôvodu radič video pamäti spotreboval približne 84 % celej blokovej SRAM pamäti. Toto rozhodnutie významne znižuje zobraziteľný počet farieb z predpokladaného počtu 2^{64} iba na 2^6 farieb. Finálny dizajn rozširujúceho modulu bude musieť byť z tohoto dôvodu pozmenený. V texte práce je popísané najflexibilnejšie riešenie s malým FPGA čipom a SRAM pamäťou priamo na rozširujúcom module.

Na komunikáciu medzi FPGA a mikrokontrolérom sa používa rozhranie Flexbus. Video pamäť alokovaná na FPGA je mapovaná do adresného priestoru mikrokontroleru. Riešenie umožňuje zapojiť k zbernici prídavné jednotky, ktoré môžu napríklad akcelerovať náročné výpočty na mikrokontroleri.

Druhá časť tejto práce sa zaoberá tvorbou GUI knižnice s podporou dotykového panelu, ktorá umožňuje zobraziť základné prvky užívateľského rozhrania. Existujúce riešenia, ktoré sú v súčastnosti dostupné sú napísané v jazyku C. Táto knižnica je experimentálne implementovaná v jazyku C++. Jazyk bol zvolený, pretože umožňuje vyššiu mieru abstrakcie

pri tvorbe programov. Architektúra knižnice je rozdelená na 3 úrovne. Implementácia je napísaná tak, aby bola užívateľsky prívetivá. Hlavnou inšpiráciou je framework Qt.

Na najnižšej úrovni sa nachádza ovládač dotykového panelu, ktorý komunikuje s dotykovým mikrokontrolerom pomocou rozhrania I2C. Druhý ovládač implementuje komunikáciu s video pamäťou pomocou zbernice FlexBus.

Na strednej úrovni sa nachádzajú pomocné funkcie, ktoré rasterizujú grafické primitíva. Knižnica podporuje vykresľovanie bodu, čiary, obdĺžniku a kruhu. Okrem toho sa na tejto úrovni spracováva orezávanie rasterizovaných primitív a maskovanie častí obrazovky. Vzhľadom na to, že sa nevyužíva dvojité bufferovanie obrazu je maskovanie potrebné pre minimalizovanie blikania obrazu.

Na najvyššej úrovni sú implementované základné prvky grafického užívateľského rozhrania, tzv. Widgety. Tlačidlo, checkbox, radiobutton a horizontal slider sú predpripravené v knižnici. Widgety sa môžu do seba zanárať. Pri zmene pozície sa všetky zanorené objekty presúvajú spolu s rodičovským elementom, pretože pozícia Widgetov je definovaná relatívne k svojmu nadradenému prvku.

Najvyššiu úroveň v topológii Widgetov tvorí okno. Knižnica podporuje zobrazenie ľubovoľného počtu okien. Správca okien zaisťuje správne prekresľovanie neplatných častí obrazu pri pohybe okien. Okrem toho zaisťuje delegáciu udalostí, ktoré vznikajú pri dotyku na dotykovom paneli. Adaptovaný je princíp callback funkcií, ktorý je typický pri paradigme „Event Driven Programming“.

Implementácia je demonštrovaná v demo-aplikácii, ktorou sa dajú riadiť LED diódy a ovládať akustický bzučiak pomocu PWM signálu. Aplikácia je dostatočne rýchla a pri zapnutí optimalizácií je beh programu niekoľko krát rýchlejší. Za povšimnutie stojí fakt, že celý kód aplikácie zaberá v pamäti približne 9 % miesta a 12 % RAM pamäti.

Modul s LCD obrazovkou významne rozširuje možnosti výukovej platformy Minerva. GUI knižnica umožňuje užívateľovi jednoducho prezentovať vlastnú aplikáciu na obrazovke. S podporou pre dotykový displej umožňuje knižnica navrhnúť aplikácie, ktoré interaktívne spolupracujú s užívateľom.

Literatúra

[1] Buchta, P. *Application Development Framework for the ARM Platform* (diplomová práce). Brno: FIT VUT v Brně, 2015. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=115265.

[2] Fakulta informačních technologií Vysokého učení technického v Brně . *MINERVA KIT* [online]. [cit. 2017-05-07]. Dostupné z: https://minerva.php5.cz/.

[3] FocalTech Systems Co., Ltd. *FT5x16 True Multi-Touch Capacitive Touch Panel Controller* [online]. 2012-04-03 [cit. 2017-01-02]. Dostupné z: http://www.newhavendisplay.com/app_notes/NV3035C.pdf.

[4] Foley, J. D. *Computer graphics : Principles and practice*. 2nd edition. Reading: Addison-Wesley Publishing Company, 1990. ISBN 0-201-12110-7.

[5] Freescale Semiconductor, Inc. *K60 Sub-Family Reference Manual* [online]. 2012-06-02 [cit. 2017-01-02]. Dostupné z: http://cache.freescale.com/files/32bit/doc/ref_manual/K60P144M100SF2V2RM.pdf.

[6] uGFX GmbH. *µGFX* [online]. [cit. 2017-05-10]. Dostupné z: https://ugfx.io.

[7] Hamsterworks Wiki. *MCB Frame Buffer* [online]. 2013-04-01 [cit. 2017-01-02]. Dostupné z: http://hamsterworks.co.nz/mediawiki/index.php/MCB_Frame_Buffer.

[8] ISSI. *IS43/46DR86400B, IS43/46DR16320B : 512Mb (x8, x16) DDR2 SDRAM* [online]. 2012-08-01 [cit. 2017-05-07]. Dostupné z: http://minerva.php5.cz/doc/datasheet/43-46DR86400B-16320B-24005.pdf.

[9] Larson, S. *VGA Controller (VHDL)* [online]. 2013-08-01 [cit. 2017-05-10]. Dostupné z: https://eewiki.net/pages/viewpage.action?pageId=15925278.

[10] Lightning, E. *µGUI* [online]. [cit. 2017-05-10]. Dostupné z: http://www.embeddedlightning.com/ugui/.

[11] MCUFont. *MCUFont : A font rendering library for microcontrollers* [online]. [cit. 2017-05-10]. Dostupné z: https://github.com/mcufont/mcufont.

[12] Newhaven Display International, Inc. *NHD-CTP6* [online]. 2016-07-27 [cit. 2017-01-02]. Dostupné z: http://www.newhavendisplay.com/specs/NHD-CTP6.pdf.

[13] Newhaven Display International, Inc. *NHD-3.5-320240MF-ATXL#-CTP-1* [online]. 2016-08-12 [cit. 2017-01-02]. Dostupné z: http://www.newhavendisplay.com/specs/NHD-3.5-320240MF-ATXL-CTP-1.pdf.

[14] Newhaven Display International, Inc. *NHD-3.5-320240MF-ATXL#-T-1* [online]. 2016-10-14 [cit. 2017-01-02]. Dostupné z: http://www.newhavendisplay.com/specs/NHD-3.5-320240MF-ATXL-T-1.pdf.

[15] NewVision Microelectronics, Inc. *NV3035C Data Sheet* [online]. 2016-03-13 [cit. 2017-01-02]. Dostupné z: http://www.newhavendisplay.com/app_notes/NV3035C.pdf.

[16] NXP® Semiconductors. *LCD Introduction to graphics and LCD technologies* [online]. 2016-03-13 [cit. 2017-01-02]. Dostupné z: http://www.nxp.com/wcm_documents/techzones/microcontrollers-techzone/Presentations/graphics.lcd.technologies.pdf.

[17] Rábová, Z.; Hanáček, P.; Peringer, P.; aj. : Užitečné rady pro psaní odborného textu. http://www.fit.vutbr.cz/info/statnice/psani_textu.html, 2008-11-01 [cit. 2008-11-28].

[18] Semtech Corporation. *SX8654/SX8655/SX8656* [online]. 2011-07-25 [cit. 2017-01-02]. Dostupné z: http://www.semtech.com/images/datasheet/sx8654.pdf.

[19] Texas Instruments Inc. *TPS61161-Q1* [online]. 2015-07-01 [cit. 2017-01-02]. Dostupné z: http://www.ti.com/lit/ds/symlink/tps61161-q1.pdf.

[20] The University of New Mexico. *Hardware Design with VHDL, Design Example: VGA* [online]. [cit. 2017-01-02]. Dostupné z: http://ece-research.unm.edu/jimp/vhdl_fpgas/slides/VGA.pdf.

[21] Xilinx, Inc. *Spartan-6 FPGA Memory Controller : User Guide* [online]. 2010-08-09 [cit. 2017-01-02]. Dostupné z: https://www.xilinx.com/support/documentation/user_guides/ug388.pdf.

[22] Xilinx, Inc. *Spartan-6 Family Overview : Product Specification* [online]. 2011-10-25 [cit. 2017-05-10]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf.

[23] Xilinx, Inc. *AR 43630 : MIG Spartan-6 MCB - PHY Calibration Steps* [online]. 2012-12-15 [cit. 2017-05-10]. Dostupné z: https://www.xilinx.com/support/answers/43630.html.

Príloha A

Obsah priloženého CD

Tabuľka A.1: Obsah CD

Cesta	Popis
`/text/pdf`	Elektronická verzia tejto práce do formáte pdf
`/text/src`	Zdrojové kódy textu tejto práce pre systém Latex
`/src/fpga/bitstream`	Bitstream súbor pre FPGA
`/src/fpga/project`	Zdrojové kódy pre projekt v Xilinx ISE
`/src/GUI library`	Zdrojové kódy demoaplikácie nad GUI knižnicou
`/module`	Zdrojové súbory k doske rozširujúceho modulu
`/doc`	Dokumentácie k súčiastkam použité v tejto práci

Príloha B

Kód demo-aplikácie

B.1 Vytvorenie okna s textom

```
/**
 * \class AboutWindow
 * \brief Show just a label with basic information to demonstrate window handler
 */
class AboutWindow : public ModalWindow
{
public:
    AboutWindow(const Rect& _position, const std::string& _title, Widget* parent,
        CallbackInstance register_s)
      : ModalWindow(_position, _title, parent, register_s)
    {
        SetOption(WindowStyleOptions::HAS_BORDER, true);
        SetOption(WindowStyleOptions::MOVEABLE, true);
        SetOption(WindowStyleOptions::HAS_TITLEBAR, true);
        SetOption(WindowStyleOptions::TITLEBAR_CLOSE, true);

        auto label = new Label(
            this, {{1, 1}, {127, 60}}, "GUI Library\n\nCreated by Oliver Nemcek\nFIT VUT 2017");
        label->SetTextColor(Colors::BLACK);
        label->SetTextAlign(Screen::TEXT_V_CENTER);
        label->SetTextAlign(Screen::TEXT_H_TOP);
        AddChild(label);
    }
};
```

B.2 Vytvorenie hlavného okna pre riadenie periférií

```
/**
 * \class TestWindow
 * \brief This is a full screen window with just one button.
```

```
 * User can display new Modal window by clicking the displayed window
 */
class TestWindow : public Window
{
public:
    TestWindow(const Rect& _position, const std::string& _title) : Window(_position, _title)
    {
        // window options
        SetOption(Window::HAS_TITLEBAR, true); // Show titlebar
        SetOption(Window::FULL_SCREEN, false); // Make it fullscreen

                // add new button to current window
                // position of window is {3, 100} and size is {160, 30}
        auto about = new Button(this, {{3, 100}, {160, 30}}, "Show About Window");
        // register callback function
        about->SetCallback(CALLBACK(TestWindow::CB_SHOW_ABOUT));
        // add button to this widget
        AddChild(about);
    }

    /**
     * \brief show new modal About window on top of this
     */
    void CB_SHOW_ABOUT(const CallbackEvent& ev)
    {
        // handle event only if button is pressed down
        if (ev.GetState() == CallbackEvent::Down)
        {
                // create new window
            auto about = new AboutWindow({{10, 10}, {130, 70}}, "About", this, {});
            // add it to window manager
            WindowManager::GetInstance().AddWindow(*about);
            // at next paint event, window will be shown
        }
    }
};
```

B.3 Main funkcia

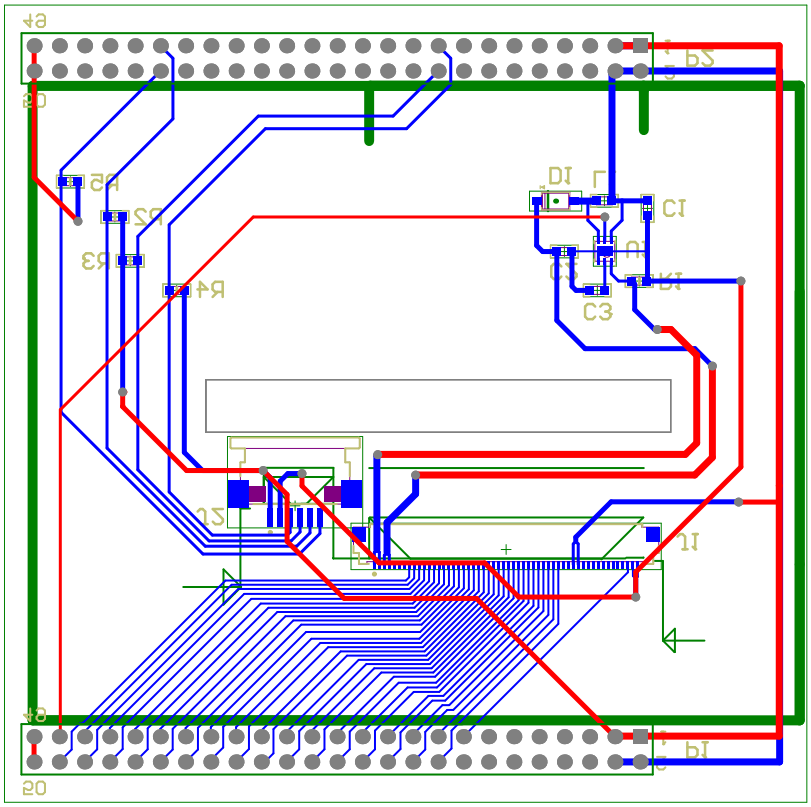
```
int main(void)
{
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    WDOG_Disable(WDOG);
    // install Systick handlers
```

```
10      Utils::InitializeSysTick();
11
12      // just some sort of UART communication
13      DbgConsole_Printf("Starting ... \r\n");
14      DbgConsole_Printf("SystemCoreClock = %lu MHz\r\n", SystemCoreClock / 1000000);
15
16      // Initialize Touch driver
17      Driver::TouchDriver t;
18      // initialize Window manager and take a reference
19      WindowManager& manager = WindowManager::GetInstance();
20
21      // this will be our test window
22      TestWindow window({{0, 0}, {320, 240}}, "Peripherals controller");
23      // add a window to window manager
24      manager.AddWindow(window);
25      // redraw screen
26      manager.Update();
27
28      // Main event loop
29      while (true)
30      {
31          // this just waits for a new touch event, but user may implement custom event loop
32          if (t.HasNewEvent())
33          {
34              // once we got new touch points
35              const TouchEvent e2 = t.GetEvent();
36              // let it pass to manager to find corrent widget
37              manager.HandleEvent(e2);
38              // and redraw screen if something has changed
39              manager.Update();
40          }
41      }
42  }
```

Príloha C

Schéma plošného spoja rozširujúceho modulu



Obr. C.1: Schéma plošného spoja